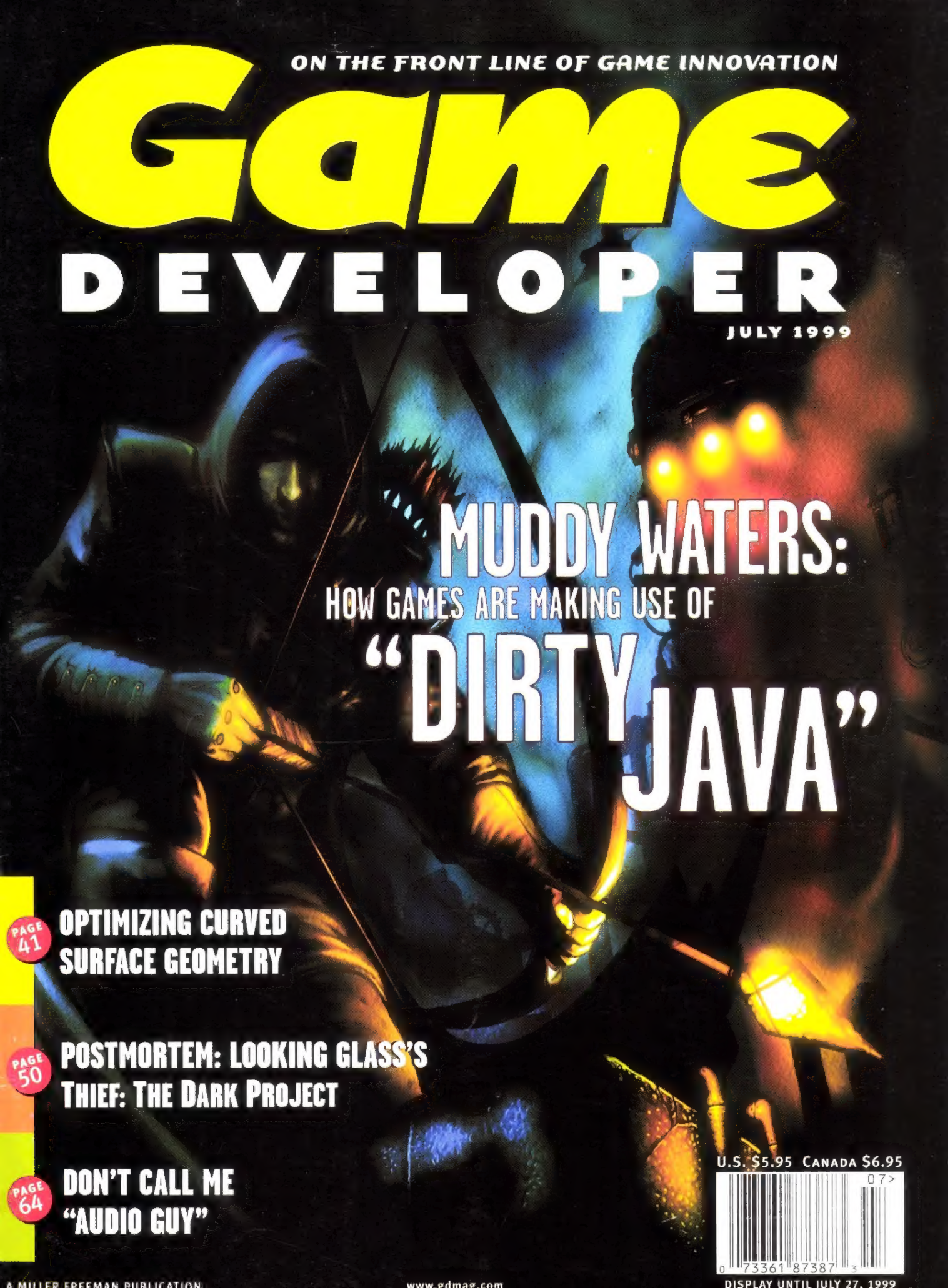
ON THE FRONT LINE OF GAME INNOVATION

Game DEVELOPER

JULY 1999



MUDDY WATERS: HOW GAMES ARE MAKING USE OF "DIRTY JAVA"

PAGE 41 OPTIMIZING CURVED SURFACE GEOMETRY

PAGE 50 POSTMORTEM: LOOKING GLASS'S THIEF: THE DARK PROJECT

PAGE 64 DON'T CALL ME "AUDIO GUY"

U.S. $5.95 CANADA $6.95

07>

0 73361 87387 3

A MILLER FREEMAN PUBLICATION

www.gdmag.com

DISPLAY UNTIL JULY 27, 1999

NETIMMERSE

Develop your game play, not your game engine.

The Fastest Path from 3D Game Conception to Completion

NETIMMERSE is a professionally engineered, C++ 3D game engine that allows you to **begin developing content from Day 1. Save 6 months of development time** and **years of engineering costs** right out of the gate. Free your software programmers for the **really cool stuff** that game players see on the screen.

Here are some of the reasons why game developers such as **Red Orb Entertainment, GreyStone Technology, Mirrormind, Mindscape, Aeon Entertainment** and others rely on NETIMMERSE:

- **It delivers action -** aggressive culling, portal-based visibility, and hand optimization in critical code
- **It delivers realism -** multi-textures, projected textures, animated textures, environment maps, skinning, and a wide variety of lighting effects
- **It delivers advanced animation -** morphing vertices, particle systems, and quaternion-based keyframes
- **It delivers scene management -** dynamic collision detection, continuous level of detail, and 3D picking
- **It's compatible -** provides a plug-in to 3D Studio MAX®, MultiGen's Creator® and Motion Factory's Motivate™, and works with Direct3D, OpenGL and Glide
- **It makes business sense -** licenses include access to full source code, no royalties, and complete support

Contact NDL today to get on the 3D game fast track!

Game Developer front line AWARD 1999

NDL

www.ndl.com

sales@ndl.com

Palo Alto, CA Sales Office - tel: 650-328-4388; fax: 650-328-3454

Chapel Hill, NC Headquarters - tel: 919-929-2917; fax: 919-967-3237

Image from *Prince of Persia® 3D*, courtesy of Red Orb Entertainment

CONTENTS

VOLUME 6, NUMBER 7

JULY 1999

FEATURES

30 Dirty Java: Using the Java Native Interface Within Games

Game developers are increasingly applying Java in "unpure" ways: using it in conjunction with C/C++ code. See how to implement dirty Java techniques, and learn how developers like id and Nihilistic applied Java to their projects.

BY BERND KREIMEIER

40 Optimizing Curved Surface Geometry

Last month we learned some fundamentals of curved surface geometry. This month Sharp explains how to optimize Bézier patch processing using a method called central differencing, which makes curved surfaces more suitable for real-time 3D graphics games.

BY BRIAN SHARP

50 Postmortem: Looking Glass's THIEF: THE DARK PROJECT

One of the most highly acclaimed titles of 1998, THIEF almost didn't make it onto shelves. Despite a fiscal crisis at Looking Glass and the loss of important team members, the team managed to stay focused and produce a game that was a success on numerous fronts.

BY TOM LEONARD

COLUMNS

12 Graphic Content — BY JEFF LANDER
Flex Your Facial Animation Muscles

20 Artist's View — BY MEL GUYMON
Talking Heads: Hierarchal Facial Animation in Real-Time 3D

26 Hard Targets — BY OMID RAHMAT
Mpath, HearMe and Mplayer

64 Soapbox — BY MATT JOHNSTON
Don't Call Me "Audio Guy"

DEPARTMENTS

2 Game Plan — BY ALEX DUNNE

5 Says You — LETTERS FROM OUR READERS

9 Bit Blasts
Alias|Wavefront rolls out Maya 2, Sega reveals pricing for Dreamcast, EA hits the billion-dollar mark, while Interplay faces red ink.

COVER: *Art by Daniel Thron, created with Fractal Design Painter 5 and Photoshop 4.*

Game Developer (ISSN 1073-922X) is published monthly by Miller Freeman Inc., 600 Harrison St., San Francisco, CA 94107, (415) 905-2200. Please direct advertising and editorial inquiries to this address. Canadian GST: 124513185. **SUBSCRIPTION RATE** Subscription rate for the U.S. is $34.95 for twelve issues. Countries outside the U.S. must be prepaid in U.S. funds drawn on a U.S. bank or via credit card. Canada/Mexico: $44.95; all other countries: $54.95 (issues shipped via air delivery). Periodical postage paid at San Francisco, CA and additional mailing offices. **POSTMASTER:** Send address changes to *Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. **Customer Service:** For subscription orders and changes of address, call toll-free in the U.S. (800) 250-2429 or fax (847) 647-5972. All other countries call (1) (847) 647-5928 or fax (1) (847) 647-5972. Send payments to *Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. For back issues write to *Game Developer*, 1601 W. 23rd St. Ste. 200, Lawrence KS 66046-2703. Call toll-free in the U.S./Canada (800) 444-4881 or fax (785) 841-2624. All other countries call (1) (785) 841-1631 or fax (1) (785) 841-2624. Please remember to indicate *Game Developer* on any correspondence.

Searching For Answers

Shortly after the horrible shooting at Columbine High School in Littleton, Colo., I picked up *The Universe and the Teacup: The Mathematics of Truth and Beauty*, by K.C. Cole. In a chapter titled "Calculated Risks," Cole contrasted societal fears about airline safety whipped up in the aftermath of the mysterious crash of TWA Flight 800 with society's seemingly resigned acceptance that thousands of children (equivalent to dozens of filled jumbo jets) die every day from malnutrition and disease around the world. She marveled at the way we tune to threats that are "exotic, personal, erratic, and dramatic." And, she noted, that doesn't mean we're ignorant, "just human."

To many, videogames appear exactly that: exotic, dramatic, and yes, dangerous. David Grossman, an Arkansas State University professor of military science, launched a crusade against violent videogames, and wants to hold them at least partially responsible for tragedies like those in Littleton and the 1997 school shootings in Paducah, Ky. He believes that games like QUAKE not only influence children, they actually train them to shoot. "A hundred things can convince someone to want to take a gun and go kill," Grossman said in prepared testimony to a Senate committee this spring. "But only one thing makes them able to kill: practice, practice, practice." The implication here is that first-person action games provide skills necessary to pull off such a shooting.

This is simplistic nonsense. It's time for pundits like Grossman to quiet down until they can submit hard evidence to support such theories. According to the *New York Times*, Mark Manes provided Eric Harris and Dylan Klebold with the TEC DC-9 used in the crime, and the three of them practiced shooting in the Colorado mountains prior to the incident. There is no evidence that any videogame trained the two killers to shoot. On the contrary, the facts seem to indicate that Harris and Klebold had real practice firing their guns.

As one peels away the layers of these cases and looks at the details as they emerge, one finds that these incidents are not as straightforward as people such as Grossman would like the public to believe. As such, I believe that going after the producers of videogames, movies, and web sites in court will prove fruitless. Holding media corporations accountable for the acts of mentally ill minors (as 14-year-old Michael Carneal pled in the Paducah case) and adults on antidepressants (as 18-year-old Eric Harris was) is no solution. Sadly, it's just another case of going after the deepest pockets.

Even if you discount any mental illness involved, will the deep pockets at companies such as Activision (one of the defendants named in the Paducah case) have to shell out? Some have pointed to the recent legal victories over tobacco companies as proof that corporations can and should be liable for the consequences of their products. But the gulf between the real dangers of cigarette smoking and the perceived dangers of playing videogames (or watching movies or surfing the web) is vast. Numerous independent medical studies have confirmed the link between smoking and cancer. Leaked internal documents and testimony from former tobacco company employees acknowledged the culpability of tobacco firms and helped seal the fate of Big Tobacco. No such evidence links videogames and violent behavior. This isn't a simple case of cause and effect.

Looking to the future, as games become more realistic and complicated, the pleasure of playing videogames may become harder to understand by those in society less attuned to this form of entertainment. They will perceive videogames as a growing threat as polygon counts grow, color depth increases, and photo-realistic scenes become normal. I don't think that videogames will be as fortunate as comic books, which were singled out in the 1950s as a harmful influence on children, and then gradually became accepted as an innocuous diversion. Videogames will always ride technology's leading edge, and as such, they're apt to leave many paranoid conservatives in their wake. As an industry, we have to understand that, and be aware that we'll be under the microscope for a long time. ■

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

600 Harrison Street, San Francisco, CA 94107
t: 415.905.2200 f: 415.905.2228 w: www.gdmag.com

Publisher
Cynthia A. Blair *cblair@mfi.com*

EDITORIAL

Editorial Director
Alex Dunne *adunne@sirius.com*
Departments Editor
Wesley Hall *whall@sirius.com*
Editorial Assistant
Jennifer Olsen *jolsen@mfi.com*
Art Director
Laura Pool *lpool@mfi.com*
Editor-At-Large
Chris Hecker *checker@d6.com*
Contributing Editors
Jeff Lander *jeffl@darwin3d.com*
Mel Guymon *mel@surreal.com*
Omid Rahmat *omid@compuserve.com*
Advisory Board
Hal Barwood LucasArts
Noah Falstein The Inspiracy
Brian Hook id Software
Susan Lee-Merrow Lucas Learning
Mark Miller Harmonix
Paul Steed id Software
Dan Teven Teven Consulting
Rob Wyatt DreamWorks Interactive

ADVERTISING SALES

Western Regional Sales Manager
Jennifer Orvik e: *jorvik@mfi.com* t: 415.905.2156
Eastern Regional Sales Manager/Recruitment
Ayrien Houchin e: *ahouchin@mfi.com* t: 415.905.2788
International Sales Representative
Breakout Marketing e: *breakout_mktg@compuserve.com*
t: +49 431 801703 f: +49 431 801797

ADVERTISING PRODUCTION

Senior Vice President/Production Andrew A. Mickus
Advertising Production Coordinator Dave Perrotti
Reprints Stella Valdez t: 916.983.6971

MILLER FREEMAN GAME GROUP MARKETING

Group Marketing Manager Gabe Zichermann
MarComm Manager Susan McDonald
Marketing Coordinator Izora Garcia de Lillard

CIRCULATION

Vice President/Circulation Jerry M. Okabe
Assistant Circulation Director Sara DeCarlo
Circulation Manager Stephanie Blake
Circulation Assistant Kausha Jackson-Crain
Newsstand Analyst Joyce Gorsuch

INTERNATIONAL LICENSING INFORMATION

Robert J. Abramson and Associates Inc.
t: 914.723.4700 f: 914.723.4722
e: *abramson@prodigy.com*

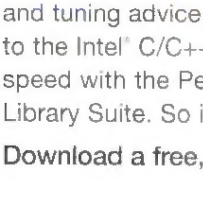
Miller Freeman
A United News & Media publication

CEO/Miller Freeman Global Tony Tillin
Chairman/Miller Freeman Inc. Marshall W. Freeman
President Donald A. Pazour
Executive Vice Presidents Darrell Denny, Galen A. Poss, Regina Starr Ridley
Sr. Vice Presidents Annie Feldman, Howard I. Hauben, Wini D. Ragus, John Pearson, Andrew A. Mickus
Sr. Vice President/Development Solutions Group KoAnn Vikören
Group President/Division SF1 Regina Ridley

BPA

www.gdmag.com

If it's not optimized
for today's processor,
it's yesterday's code.



Optimize your code for the latest Intel® processor with the people who know it best.

Introducing the award-winning VTune™ Performance Enhancement Environment 4.0, the newest comprehensive optimizing solution from Intel. Now programmers can take full advantage of the capabilities of today's Intel® Pentium® III processor. The VTune Performance Analyzer, included in the package, provides invaluable software analysis and tuning advice. Plus, the VTune Performance Enhancement Environment includes a powerful upgrade to the Intel® C/C++* and Fortran* Compiler, Intel's own Performance Training Center to help get you up to speed with the Pentium III processor, and new versions of the Intel® Performance Library Suite. So if you want today's code, optimize it for today's Intel® processor.

Download a free, 14-day evaluation at: http://developer.intel.com/vtune

intel®

© 1999 Intel Corporation. All rights reserved. Intel and Pentium are registered trademarks, and VTune is a trademark of Intel Corporation. *All other brands and trade names are the property of their respective owners.

Play back up to sixteen 24-bit digital tracks simultaneously, mix with the power of a Yamaha 02R and add professional 32-bit effects and EQ without ever letting your computer know it's working.

The Yamaha DSP Factory gives you a constant supply of audio production firepower. The PCI card includes Yamaha-designed chips with plenty of muscle to handle all of your recording, mixing, dynamics and effects processing needs. In other words, your computer's processor never limits the DSP Factory's extensive capabilities; all it has to do is run your favorite recording software.

The DSP Factory comes with the DS2416 PCI card and stereo analog and digital in/out jacks. Expand the system to fit your specific needs with the AX44 or the new AX88, which gives you 8 channels of 24-bit analog I/O. However you configure it, **Recording** magazine says you'll get a "serious studio in a box." And **Electronic Musician** says "the processing is top-notch, the mixer and recorder are extremely powerful...and the whole thing sounds great." Just don't tell your computer it's really a music studio.

YAMAHA

AdB PERFECT SOUND

For more information, visit www.yamaha.com/proaudio or call (800)937-7171 ext. 632 for literature.

©1999, Yamaha Corporation of America, Pro Audio Department, P.O. Box 6600, Buena Park, CA 90622-6600

Does Sex Sell?

I find your magazine to have way too many sexual connotations. Are these adult games only or do they market to children, too? Advertisements on pages 18, 19, 28 and 63 (April 1999) are very inappropriate. I understand that sex sells, but isn't there another way companies can think of to make a buck and save our society and children from negative, wasted energy? Basically, I was disgusted by the severity of sexuality towards women.

Jennifer Dennis
via e-mail

I must protest the increasing number of advertisements that feature scantily-clad women and sexual innuendo appearing in *Game Developer*. Looking back over the last several issues, the number seems to have jumped sharply in April.

The game industry has long had a reputation as a boys' club which is hostile to women. This reputation has compromised our ability to attract top female talent to our companies, and therefore, to make the best games possible. Many women developers are uncomfortable with the blatant appeals to sexuality which are the stock-in-trade of our industry's advertising. This material now seems to be creeping from the gamers' magazines into our own trade journals, and it sends a distinct message that women are not wanted or welcome in this business. I trust this is not a message that *Game Developer* agrees with.

These advertisements are insulting to our intelligence as developers. No developer is so foolish as to make technical purchasing decisions on the basis of whose ads feature nude women; and to assume that we would is to characterize us as oversexed and stupid.

As a longtime subscriber to a variety of trade journals, I assure you that such material is neither ordinary nor appropriate for them. It certainly does not appear in *Electronic Engineering Times*, for example, even though its readership, like *Game Developer*'s, is also predominantly male. I urge you to reject any more advertising on these themes.

Ernest W. Adams
Electronic Arts
Redwood City, Calif.

Really, We're Just Friends

Your article "Dolby and Aureal: Contrasts in Audio" (Hard Targets, May 1999) was interesting and insightful, and I thank you for writing it.

I am writing to correct you on one point you made near the end of the article: "Dolby then has to look at DTS, and George Lucas's THX among other competitors." The point I wish to correct is regarding THX as a competitor to Dolby. This is a common misconception that THX and Dolby are competing technologies, when in fact they complement each other.

Bone to pick? E-mail adunne@sirius.com. Or write to *Game Developer*, 600 Harrison Street, San Francisco, CA 94107.

THX is a hardware certification program to ensure the best sound reproduction, in other words, to reproduce it as close as possible to the original. Dolby SR, Dolby Digital, DTS, and SDDS are sound formats that, if reproduced on the highest-quality THX-certified systems, will produce an unsurpassed quality of audio. You are correct when you state DTS as competition to Dolby.

Paul Widner
via e-mail

Let Him Who Has Played Sin Cast the First Stone

I had particular interest in your Postmortem on Ritual Entertainment's SIN (March 1999). Indeed, the column is my favorite in the magazine, and I was well aware of the horrible bugs in the initial release of SIN.

In reading the article, I was a little shocked that so little mention was made of the bugs on release. At the very least, they should have been under one of the "Things That Went Wrong." There are a few issues surrounding author Scott Alden's claims that disturb me.

First, just how bad is their quality assurance group? By some reports, the game took approximately five minutes to load both between levels and when restoring a saved game. If the game was run from the CD auto-play screen, it ran from the CD instead of the hard drive, causing save game troubles. Saved games themselves were enormous — people were losing hundreds of megabytes from them! There was also a bug with the first boss encountered, which ruined the encounter entirely for many people. These bugs simply "slipped through"?

Second, there was mention of the patch, and Activision distributing free CDs to all who requested them. I do not feel that was an act of charity, but an attempt to right a wrong. Some of the newsgroup chatter hinted at in the article mentioned an apparent release agreement with Activision. If the game was not out by a certain date, Ritual was to suffer a rather large economic penalty. (Presumably, the intention was to beat HALF-LIFE to market.) Hence, the game shipped early with known bugs.

Hearsay? Conjecture? Or swept under the carpet?

Jose Fernandez
via e-mail

RITUAL'S LEVELORD RESPONDS:

First, you must realize that no game developer ever releases bugs intentionally. In the case of SIN, we worked very long and hard for almost two years. It seemed to be an endless crunch time and I nearly had a nervous breakdown from the long-term stress and time demands. I know this seems like a dream job, and it is, but I truly almost lost my mind from SIN. Losing your mind, by the way, is not fun like some acid trip — it's very scary!

Anyhow, you must know that we wouldn't just throw something out after all that work. If it means anything to say this, we are heartbroken that such a cool game as SIN is now used as a symbol of what not to do. There are other reasons for SIN's demise.

Similar to proofreading your own writing after you've already read it a thousand times, or taste-testing a meal after you've been in the kitchen all day nibbling the ingredients

and smelling the seasonings, you really can't effectively and thoroughly verify your own work. You start to see the forest and not the trees, and you make too many assumptions in the fog of familiarity. You need new eyes to truly test anything. We were counting on our publisher to do this.

Being a small development team compared to most out there, we also lacked the person-power to perform complete testing. We were counting on our publisher to supplement this need. Each tribe member was hurriedly rushing right up until the very last moment. Burned out and fatigued, it was like trying to herd a dozen cats after running a marathon. Remember the scene in Jaws *in which Richard Dreyfuss is frantically tying a buoy to the spear gun line and yelling at Robert Shawn, "Don't wait for me!" as he aimed at the approaching shark? That's what it was like at the end of* Sin. *That's what the end of most games is like.*

We were assured that a large team of testers would verify the game and make sure it was valid. When we went beta, we indeed got a large load of bugs and misbehaviors. This list, however, never seemed to change much other than to get smaller as we checked off each one. This seems obvious now and a warning flag should have gone up, but we were too concentrated on finishing and burned out from exhaustion. It all ended with a "No show-stoppers! We're going gold!" from the publisher. We assumed.

Being small also means that we do not have all the varieties of test platforms (sound cards, video cards, and so on) to test across the board. We also tend to develop on our network where slow load times are the norm due to pipeline traffic. We simply do not have the time to burn CDs and load the game on isolated platforms, especially towards the end. We were counting on our publisher to do the hardware-related testing, too.

Please know that we tried very, very hard to make Sin *the coolest game of 1998. It was the coolest, if not a good second place. Also know that we did everything possible to debug the final version and test every nuance and permutation of game play. Finally, know that no one is hurt more by this rush-to-the-shelves calamity than we are. After having the privilege of working on such excellent games as* Duke Nukem 3D *and* The Scourge of Armagon, *it was a tremendous blow that* Sin *was not to follow suit merely for reasons of bad management and handling.*

Should the Hall of Fame Be the Hall of Games?

It's revealing sometimes to glimpse at what makes people do the things they do. Often it's a complex combination of physical, psychological, and yes, spiritual impulses. For some reason, we in the entertainment industry seem to want to reveal those driving influences more than the average person does. Maybe it's because we're in the business of having fun and we think that others might want to know why we are in this crazy business. Sometimes they really do want to know. But I believe that mostly they don't.

It's with these thoughts in mind that I consider Ernest Adams's notion of a Computer Game Hall of Fame ("Immortality for Game Developers," Soapbox, April 1999). While the concept of a Hall of Fame is intriguing, he told me way more than I wanted to know. For Ernest, it's all about immortality. He wants his own pyramid. Well I have news for you, Ernest. The pyramids may seem immortal, but the pharaohs never do.

Computer gaming is a relatively young endeavor compared to other pursuits. I like the fact that we don't have a lot of self-appointed experts in tweed coats rubbing their chins and harrumphing about this or that "important" game in the history of games. Adams states that the Hall of Fame "would be a place where the great games are kept, and talked about, and studied for the wonder and truth that they contain. Above all, it would be a place where their designers are honored." Wonder and truth? The fact is that the games we make are fun and technically amazing and yes, worthwhile. But let us not get carried away and overstate our contribution to mankind.

Having a place where we can see the development of computer games — to see the history of computer gaming including the designers — is a good and interesting idea. A place where we mostly raise monuments to the builders is less satisfying. I understand that Mr. Adams lost a friend and I understand wanting to honor that friend. I also understand the need for recognition and love. But I would prefer to honor the achievement much more than the achievers. After all, as Shakespeare might have said, the game's the thing.

Glenn O'Bannon
Rainbow Studios
via e-mail

In response to Ernest Adams's Soapbox column "Immortality for Game Developers" (April 1999), computer game museums do exist. Look up http://www.computerspielemuseum.de and http://www.trans-japan.com/vp/bg90.

But the idea is nonsense. Art that cannot be experienced is void. We don't need a museum; we need good emulators, and we need the greedhead companies that crack down on them to figure out a way to let them thrive. Dani Bunten will be remembered as more than a marginal figure only if future generations can experience her games.

Greg Costikyan
via e-mail

ERNEST ADAMS RESPONDS:

"L'homme n'est rien; c'est l'oeuvre qui est tout," *as Gustave Flaubert observed to George Sand. ("The man is nothing; the work is everything.") Ignoring the author in favor of the work is the* modus operandi *of oh-so-trendy postmodern literary criticism, and Glenn O'Bannon — his derision for intellectuals aside — places himself squarely in that camp. Call me old-fashioned, then, but I must disagree. To honor the artist along with the art is no more than simple justice, as Greg Costikyan eloquently argues in his online essay on the subject, which can be found at http://www.crossover.com/costik/justice.html. Should we remember Shakespeare's plays, but forget Shakespeare? Remember Mozart's music, but forget Mozart? Remember Spielberg's movies, but forget Spielberg? That would be cruelty, indeed.*

If those figures are too grand, then try considering my original model, the Pro Football Hall of Fame. Can we remember Walter Payton's running without remembering Walter Payton, or Dick Butkus's tackling without remembering Dick Butkus? Should we try? The best computer games are expressions of a guiding vision. Let us then praise the visionaries along with their work.

Do you lust for specs like these?
32
BIT COLOR RENDERING
128
BIT TECHNOLOGY
32
MB MEMORY
GRAPHICS BY
rage 128
The Fastest True Color Gaming Platform
"It delivers fast frame rates, beautiful images, 32-bit rendering performance, a whopping 32MB memory, TV out, and DVD decoding in a cool, inexpensive package."
Marcia & Barry Press, Dec. 1998, www.avault.com
"The RAGE 128 can render at 32-bit color depth without the 20-50% performance decrease all other chipsets come with."
Anand Lal Shimpi, Dec. 1998, www.anandtech.com
"The power of the RAGE FURY shines through helped by its 32MB on board RAM."
Kyle Strasbourg, Dec. 1998, www.cybertitan.com
"Rage Fury fires out a strong 3D Mark 99 rating, especially at higher color depths."
Craig Campanaro, Dec. 1998, www.sharkyextreme.com
THE POWER OF SPEED
ATi
SATISFY YOUR LUST AT WWW.RAGECONSPIRACY.COM
http://www.atitech.com/ad/gd
© Copyright 1999, ATI Technologies Inc. ATI, RAGE 128 and RAGE FURY are trademarks and/or registered trademarks of ATI Technologies Inc.

RESOLUTION MATTERS

COLOR RESOLUTION PIXEL RESOLUTION FRAME RATE

Does your video compression software leave you feeling unsatisfied?

INTRODUCING TRUEMOTION® 2X

the newest addition to the TrueMotion® family of codecs from The Duck Corporation

DIRECTSHOW QUICKTIME WIN32 WINCE MACINTOSH DREAMCAST TRUEPLAY STANDALONE SDK

TrueMotion 2X is a 3rd generation true-color codec with full YUV16 resolution, up to twice the performance of TrueMotion 2.0, plug-ins for all major platforms, TruePlay SDK with powerful "C" API for custom integration, flexible licensing terms with no per-disk royalties for PC based titles. You don't have to skimp on quality or performance. Whether your application calls for CD-ROM playback or super high resolution video, TrueMotion 2X can handle the job.

Download a free evaluation copy today.

BIT Blasts

New Products: Sven Technologies updates SurfaceSuite Pro, Alias Wavefront rolls out Maya 2, and OpenGl arrives for Macintosh **p. 9**

Industry Watch: Sega reveals pricing for Dreamcast, EA hits the billion dollar mark, Activision squeaks by, and Interplay faces red ink. **p. 10**

New Products

by Alex Dunne

SurfaceSuite Pro 1.5

SVEN TECHNOLOGIES released version 1.5 of SurfaceSuite Pro, the company's 3D texture mapping software which lets you apply 2D images to 3D polygonal, NURBS and patch models. Version 1.5 adds two significant new features to the product. The first, AlphaPaint, lets you paint alpha masks onto your textured model, giving you more control over texture blending. The second addition is PatchWork, which lets you create a "quilt" of textures — a single skin — for real time polygonal models that you can export to your game engine.

The new version also supports Softimage's .HRC and Rhino's 3D model formats, as well as the .PCX image format. Sven perked up the product's interface by adding multiple keyboard shortcuts and hotkeys, floating windows, automatic layer creation, and better support for importing materials.

SurfaceSuite Pro supports 3D Studio Max, Softimage, Maya, LightWave 3D, and Rhino, among other 3D modeling/animation environments. The standalone product is priced at $595, and upgrade discounts are offered.

■ Sven Technologies
Palo Alto, Calif.
(650) 852-9242
www.sven-tech.com

Maya 2

ALIAS|WAVEFRONT introduced Maya 2, the latest version of its 3D animation and visual effects tool. Perhaps most important to game developers, the company improved the modeling tools — you now have more precise control over curve and surface geometry, new tools for smoothing polygonal surfaces new texturing tools and the ability to assign arbitrary data to polygon vertices. With the Maya Unlimited package, you get more advanced modeling tools (including support for subdivision surfaces) as well as fur and cloth animation features.

Maya 2's enhancements promise game developers improved animation features, including new fur rendering, as seen on this tiger.

In the area of character modeling and animation, Maya 2 has new deformer types and automated skinning capabilities for faster creation and easier control of complex characters. The tool uses a pose-based approach that lets you treat a complex character as a single entity as you animate it. Alias|Wavefront also souped up Maya's renderer, which can speed rendering up by 90 percent in some cases, according to the company. Maya 2 also supports multithreaded batch rendering, too.

Maya Complete 2 is priced at $7,500 and includes modeling, rendering, animation, dynamics, Artisan and MEL (the tool's embedded scripting language) features. Maya Unlimited 2 costs $16,000 and includes the features found in the Complete version, plus Maya Live, Maya Fur, Maya Cloth and new advanced modeling features. Both Maya flavors are available for Windows NT and IRIX.

■ Alias|Wavefront
Toronto, Ont., Canada
(416) 362-9181

OpenGL for the Macintosh

APPLE recently shipped OpenGL for Macintosh, further illustrating the way in which Steve Jobs is shifting Apple's focus back towards games. Available freely from the Apple web site for download, OpenGL for Macintosh brings the industry-standard 3D API to the Mac. OpenGL for Macintosh requires a PowerPC-based Macintosh computer with MacOS 8.1 or later. An iMac or new Power Macintosh G3 is recommended for accelerated 3D rendering. The version available from Apple's web site includes libraries to accelerate rendering on Rage II, Rage Pro, and Rage 128-based Macintosh systems.

■ Apple Computer
Cupertino, Calif.
(408) 996-1010

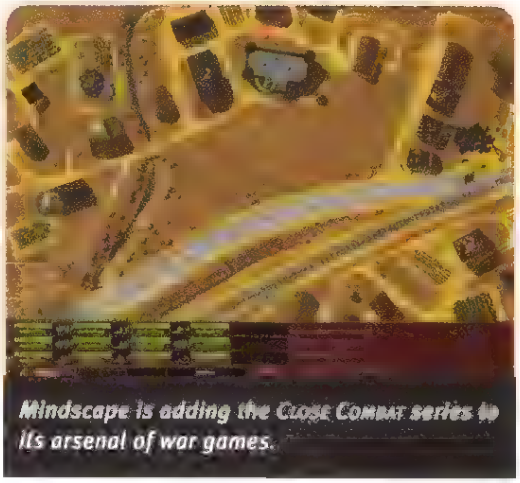
Industry Watch

by Alex Dunne

DISNEY INTERACTIVE AND NINTENDO announced a new partnership, which will introduce Mickey Mouse to the world of 3D games. Nintendo will publish a number of Mickey Mouse-based games for the N64 and the Game Boy Color, to be developed by Rare Ltd. Two of the titles will be racing games for the N64 and Game Boy Color (coming out in 1999 and 2000), and the third will be a "Mickey Adventure" title, slated for release in 2001 on Nintendo's next console system and the Game Boy Color. Additionally, as part of the agreement, Disney Interactive will develop multiple titles for the Game Boy Color aimed at the girl-games market. These titles will be based upon Disney's *Beauty and the Beast* and *Alice in Wonderland* movies.

SPEAKING OF THE GAME BOY COLOR, Nintendo said that sales of the Game Boy Color during the first quarter of 1999 averaged 94,000 units per week, compared to 31,000 in 1998 for the previous Game Boy unit. Peter Main, Nintendo's executive vice president of sales and marketing, said that since the launch of the Game Boy Color in the U.S. last November, the company has sold more than two million units.

ATOMIC POWER AT MINDSCAPE. Mindscape signed a publishing deal with Atomic Games, which lets the 'Scape take over the successful CLOSE COMBAT series from former publisher Microsoft. The deal adds another strong war game franchise to Mindscape's lineup, which already included PANZER GENERAL and STEEL PANTHERS. The next game scheduled for the CLOSE COMBAT series is slated for release sometime in the fourth quarter.



Mindscape is adding the CLOSE COMBAT series to its arsenal of war games.

ACTIVISION BARELY BEATS ESTIMATES. Activision's fourth fiscal quarter (ending March 31) profits were a bit better than analysts had expected, amounting to $5.2 million, which was an improvement over the $689,000 that the company earned last year during the same period. CEO Bobby Kotick credited the company's wide array of games for boosting the company's market position, and indicated that the company was on the prowl for large acquisition targets. However, CFO Barry Plaga predicted Activision would post a loss in the first fiscal quarter of 1999 as it did last year.

ARCADE DOWSER. In an attempt to counteract the shrinking arcade market and make it easier to bag that increasingly elusive quarry, the loyal arcade gamer, Midway Games and WMS Industries launched a web based database application that tells you exactly where you can find Midway coin-op games around the world. This simple game finder, available on on Midway's website ( ), also lets visitors submit new locations when they find games. Planning a trip to Madagascar? Need to satisfy your CARNEVIL fix while you're there? Now you know where go...

MUSIC TO THEIR EARS? Aureal Semiconductor, maker of the Vortex2 digital audio processor and the A3D audio API, announced financial results for the first fiscal quarter ending April 4, 1999. Revenues reached $12.6 million, amounting to a 250 percent increase, but saw a net loss of $4.1 million (better than its $5.5 million loss last year). What looks most promising is that the company's gross margins grew to 34 percent in the first quarter from 22 percent last year, thanks to increased demand for sound cards based on the Vortex2.

DREAMCAST PRICING AT LAST. Undercutting industry estimates, Sega's Bernie Stolar revealed that the the Dreamcast suggested launch price would be $199. Additionally, Stolar confirmed that the U.S. launch date would be September 9, 1999. There will be between 10 and 12 titles available around launch time, plus more than 20 first-party titles on track for the year 2000. To date, retailers have placed pre-orders for 30,000 systems, according to Sega.

$109. EA released its fiscal year results. Two words: banner year. Congratulations go out to Larry Probst and company, for being the first entertainment software publisher to top a billion dollars in revenue.

NEWS FROM THE LAND OF RED INK. Poor Interplay continues to have a tough time. The company lost $8.28 million in its first quarter, compared to a profit of $3.1 million a year ago. Interplay blamed its larger-than-expected loss on the fact that it hasn't released any major titles recently, and the impact of higher-than-anticipated product returns. ■

UPCOMING EVENTS CALENDAR

Macworld Expo

JACOB J. JAVITS CONVENTION CENTER
New York, NY
July 21–23, 1999
Cost: $45–$1,195
http: www.ma

Siggraph '99

LOS ANGELES CONVENTION CENTER
Los Angeles, Calif.
August 8–13, 1999
Cost: $25–$760
http: www.siggraph.org

Of course it helps you work smarter. It has four

brains

Do more in less time with the only four-processor graphics workstation for Windows NT.

The new Silicon Graphics 540™ visual workstation is the ultimate tool for working smarter—not harder. Now you can [illegible] without the usual slowdown. [illegible] breakthrough multiprocessing [illegible] Advanced Graphics Architecture delivers the performance [illegible] Windows NT. [illegible] before. And the [illegible] Gateway 540 is an open architecture product that [illegible]

Silicon Graphics 540	Single Intel® Pentium® III Xeon™ processor 500MHz	$6,495

- [illegible]
- [illegible]
- [illegible]
- Integrated 10/100 Fast Ethernet, IEEE [illegible]
- [illegible]
- [illegible]

intel inside
pentium® III xeon™

[illegible] 1 888 SGI-7509 or visit us at www.sgi.com/go/visual

PCWEEK BEST OF COMDEX

[illegible]

sgi
The solution is in sight

by Jeff Lander

Flex Your Facial Animation Muscles

Last month I left off with a nice short list of the visemes I would need to represent speech realistically. However, now I am left with the not insignificant problem of determining exactly how to display these visemes in a real-time application.

It may seem as if this is purely an art problem, better left to your art staff (see Artist's View this month, "Talking Heads: Hierarchical Animation in Real-time 3D"). Or, if you are a one-person development team, at least left to the creative side of your brain. However, your analytical side needs to inject itself in here a bit. This is one of those early production decisions you read about so much in the Postmortem column that can make or break your schedule and budget. Choose wisely and everything will work out great. Choose poorly and your art staff or even your own brain will throttle you.

Decisions, Decisions

For the final result, I want a 3D real-time character that can deliver various pieces of dialog in the most convincing manner possible. Thanks to the information learned last month, I know I can severely limit the amount of work I need to do. I know that with 13 visemes, or visual phoneme positions, I can reasonably represent most sounds I expect to encounter. I even have a nice mapping from American English to my set of visemes. Most other languages could probably be represented by these visemes as well, but could require a different mapping table.

From this information I can expect that if I can reasonably represent these 13 visemes with my character mesh, then continuous lip-synch should be possible. So the problem really comes down to how I construct and manipulate those meshes.

Viseme-Based Methods

Certainly, the obvious method for creating these 13 visemes is to generate 13 versions of my character head mesh, one to represent each viseme. I can then use the morphing techniques I discussed in last December's column ("Mighty Morphing Mesh Machine," December 1998) to interpolate smoothly between different sounds.

Modeling the face to match the visemes is pretty easy. Once the artist has the base mesh created, each viseme can be generated by deforming the mesh any way necessary to get the right target frame. As long as no vertices are added or deleted and the triangle topology remains the same, everything should work out great. Figure 1 shows an image of a character displaying the "L" viseme, as in the word "life." The tongue is behind the top teeth, slightly cupped, leaving gaps at the side of the mouth, and the teeth are slightly parted.

Sounds pretty good so far. Just create 13 morph targets for the visemes in addition to the base frame and you're done. Life's great, back to physics, right? Well, not quite yet.

Suppose in addition to simply lip-synching dialog, your characters must express some emotion. You want them to be able to say things sadly, or speak cheerfully. We need to add an emotional component to the system.

Adding Some Heart to the Story

At first glance, it may seem that you can simply add some additional morph targets for the base emotions. Most people describe six basic emotions. Here they are with some of their traits. (See Goldfinger under "For Further Info" for photo examples of the six emotions.)

1. **Happiness:** Mouth smiles open or closed, cheeks puff, eyes narrow.
2. **Sadness:** Mouth corners pull down, brows incline, upper eyelids droop.
3. **Surprise:** Brows raise up and arch, upper eyelids raise, jaw drops.
4. **Fear:** Brows raise and draw together, upper eyelids raise, lower eyelids tense upwards, jaw drops, mouth corners go out and down.
5. **Anger:** Inner brows pull together and down, upper eyelids raise, nostrils may flare, lips are closed tightly or open exposing teeth.
6. **Disgust:** Middle portion of upper lip pulls up exposing teeth, inner brows pull together and down, nose wrinkles.

There are variations of these emotions, such as contempt, pain, distress, excitement, but you get the idea. Very distinct versions of these six will get the message across.

The key thing to notice about this list is that many of these emotions directly affect the same regions of the model as the visemes. If you simply

When not massaging the faces of digital beauties or doing stunt falls in a mo-cap rig, Jeff can be found flapping his own lips at Darwin 3D. Send him some snappier dialogue at jeffl@darwin3d.com.

Wicked

NT performance.
Wicked pricing.

PCWEEK COMDEX

Silicon Graphics 320™ — $3,695

- [illegible]

Silicon Graphics 540™ — $6,495

- [illegible]

Silicon Graphics 320 — $4,684

- [illegible]

Silicon Graphics 540 — $7,313

- [illegible]

Silicon Graphics 320 — $5,654

- [illegible]
- *Upgrade to the Silicon Graphics 1600SW 17" digital flat panel monitor for $2,044*

Silicon Graphics 540 — $9,952

- [illegible]
- [illegible]

intel inside
pentium® III

To get information, find a local reseller or to order call **1 888 SGI-7509** or visit us at **www.sgi.com/go/visual**

[illegible]

sgi
The solution is in sight

FIGURE 1. *The "I" viseme as seen at the start of the word "life."*

FIGURE 2. *A very surprised "I" viseme.*

layer these emotions on top of the existing viseme morph targets, you can get an additive effect. This can lead to ugly results.

For example, let me start with the "L" sound from before and blend in a surprised emotion at 100 percent. The "L" sound moves the tongue up to the top set of teeth and parts the mouth slightly. However, the surprise target drops the jaw even farther but leaves the tongue alone. This combination blends into the odd-looking character you see in Figure 2.

This problem really becomes apparent when the two meshes are actually fighting each other. For example, the "oo" viseme drives the lips into a tight, pursed shape while the surprise emotion drives the lips apart. Nothing pretty or realistic will come out of that combination.

When I ran into this issue a couple of years ago, the solution was tied to the weighting. By assigning a weight or priority to each morph target, I can compensate for these problems. I give the "oo" viseme priority over the surprise frame. This will suppress the effect that the surprise emotion has over shared vertices.

Welcome to Muscle Beach

Most of the academic research on facial animation has not approached the problem from a viseme basis. This is due to a fundamental drawback to the viseme frame based approach. In the viseme-based system, every source frame of animation is completely specified. While I can specify the amount each frame contributes to the final model, I cannot create new source models dynamically. Say, for example, I want to allow the character to raise one eyebrow. With the frames I have described so far, this would not be possible. In order to accomplish this goal, I would need to create individual morph targets with each eyebrow raised individually. Since a viseme can incorporate a combination of many facial actions, isolating these actions can lead to an explosive need for source meshes. You may find yourself breaking these targets into isolated regions of the face.

For this reason, researchers such as Frederic Parke and Keith Waters began examining how the face actually works biologically. By examining the muscle structure underneath the skin, a parametric representation of the face became possible. In fact, psychologists Paul Ekman and Wallace Friesden developed a system to determine emotional state based on the measurement of individual muscle groups as "action units." Their system, called Facial Action Coding System (FACS), describes 50 of these action units that can create thousands of facial expressions. By creating a facial model that is controlled via these action units, Waters was able to simulate the effect that changes in the action units reveal on the skin.

While I'm not sure if artists are ready to start creating parametric models controlled by virtual muscles, there are definitely some lessons to be learned here. With this system, it's possible to describe any facial expression using these 50 parameters. It also completely avoids the additive morph problem I ran into with the viseme system. Once a muscle is completely contracted, it cannot contract any further. This limits the expression to ones that are at least physically possible.

Artist-Driven Muscle-Based Facial Animation

Animation tools are not really developed to a point where artists can place virtual muscles and attach them to a model. This would require a serious custom application that the artists may be reluctant even to use. However, that doesn't mean that these

Action	Muscle Name	Effect
Raise Inside Brow L/R	*Frontalis Medial portion*	Many Expressions
Raise Outside Brow L/R	*Frontalis Lateral portion*	Many Expressions
Tighten Inside Brow Frown	*Corrugator Supercilii + Procerus*	Anger, Pain, Disgust
Eyes Wide L/R	*Levator Palpebrae Superioris*	Surprise, Fear, Shock
Eye Squint L/R	*Orbicularis Oculi orbital portion*	Anger, Thought, Concentration
Eyelid Close L/R	*Orbicularis Oculi palpebral portion*	Blink, Wink
Nostril Flare L/R	*Dilator Naris + Levator Labii Superioris Alaeque Nasi*	Disgust
Purse Lips	*Incisivus Labii*	Kiss, Anger, "oo", Whistle
Smile Corner L/R	*Zygomaticus Major*	Smile
Corner mouth down into Sadness L/R	*Depressor Anguli Oris + Zygomaticus Minor + Depressor Anguli Oris + Mentalis*	Sadness
Top Lip Up L/R	*Levator Labii Superioris*	Disgust, Part Lips for Sounds
Lower Lip Down L/R	*Depressor Labii Inferioris*	Part Lips for Sounds
Tighten Lips L/L	*Orbicularis Oris*	"p", "b", "m", Anger
Jaw Open	*Digastric*	Speaking, Surprise
Jaw Slide L/R	*Masseter*	Slide Jaw L/R

CHART 1. *The basic muscle groups involved in facial animation.*

MultiGen Creator

MultiGen Creator for Games
Now only $3500*

The Complete Realtime 3D Toolset

Join The Hitmakers

A partial list of hit titles built using MultiGen-Paradigm tools:

Atari Games/Midway
[illegible] RUSH™ (N64, Arcade, PC, PC/)

Rare Ltd.
GoldenEye 007™ (N64)
Banjo Kazooie™ (N64)
Twelve Tales: Conker 64™ (N64)

Zipper/EA/Hasbro
MechWarrior III™ (PC)
Top Gun: Hornet's Nest™ (PC)
Recoil™ (PC)

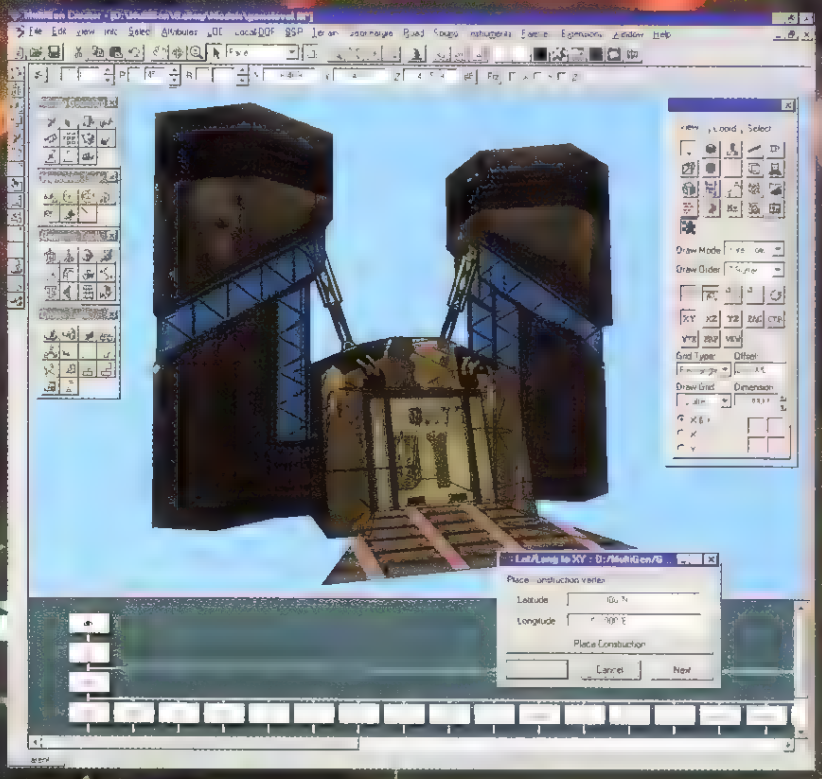
BlueShift, Inc.
Vapor TRX™ (Arcade) [illegible]

Nintendo
Super Mario™ (N64)
Zelda™ (N64)

Origin/EA
Longbow II™ (PC)
Jane's F-15™ (PC)

Sierra On-Line, Inc.
Viper Racing™ (PC)

SingleTrac
Twisted Metal™ I, II & III (PSX & PC)
Jet Moto™ I & II (N64)
Warhawk™ (PSX)



HIGHLIGHTS OF MULTIGEN POWERTOOLS

- Import Art from 3D animation tools
  - 3D Studio MAX
  - SoftImage
  - Alias
  - Nichimen . . . Many More
- Game Data Tagging and Editing
- Automated Level of Detail (LOD)
- Hierarchy Structure Editing
- Binary Separating Planes (BSP)
- Precise UV Texture Mapping/Editing
- Texture Rendering Control
- Damage and Switch States
- Degrees of Freedom (DOF)
- Collision and Bounding Volumes
- Instancing and External Referencing
- Automated Terrain
- Automated Roads and Pathways
- Scene Optimization
- World and Level Assembly
- Compatable with Multiple Realtime Formats
  - Openflight
  - VRML
  - NIFF
  - HMD
- Extensibility & Customization
  - Data Read/Write API for converters and loaders
  - Data Extensions API for custom game data
  - Plug-in API for custom tools
- Multimedia Asset Management

Check Out MultiGen Creator

Contact us to request a free evaluation of MultiGen Creator. Visit the MultiGen-Paradigm web site to download the OpenFlight file format and get lots of useful realtime information

Contact us for more details
Phone: 408 261 4100
Fax: 408 261 4103
email: sales@multigen.com
http://www.multigen-paradigm.com

* Must be a licensed Sony or Nintendo Developer
* Exports only Niff or HMD formats

Images from Recoil™ courtesy Electronic Arts & Zipper Interactive, Inc.

MultiGen Paradigm

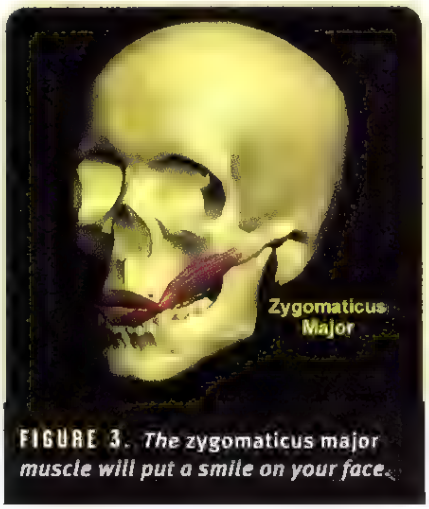


FIGURE 3. *The zygomaticus major muscle will put a smile on your face.*

FIGURE 4. *Pucker up:* Incisivus labii *at work.*

methods are not available for game production. It just requires a different way of thinking about modeling.

For instance, let me take a look at creating a simple smile. Biologically, I smile by contracting the *zygomaticus major* muscle on each side of my face. This muscle connects the outside of the zygomatic bone to the corner of the mouth as shown in Figure 3. Contract one muscle and half a smile is born.

O.K. Mr. Science, what does that have to do with modeling? Well, this muscle contracts in a linear fashion. Take a neutral mouth and deform it as you would when the left *zygomaticus major* is contracted. This mesh can be used to create a delta table for all vertices that change. Repeat this process for all the muscles you wish to simulate and you have all the data you need to start making faces. You will find that you probably don't need all 50 muscle groups described in the FACS system. Particularly if your model has a low polygon count, this will be overkill. The point is to create the muscle frames necessary to create all the visemes and emotions you will need, plus any additional flexibility you want. You will probably want to add some eye blinks, perhaps some eye shifts, and tongue movement to make the simulation more realistic.

The FACS system is a scientifically-based general modeling system. It does not consider the individual features of a particular model. By allowing the modeler to deform the mesh for the muscles instead of using this algorithmic system, I am giving up general flexibility over a variety of meshes. However, I gain creative control by allowing for exaggeration as well as artistic judgement.

The downside is that it is now much harder to describe to the artists what it is you need. You need to purchase some sort of anatomy book (see my suggestions at the end of the column) and figure out exactly what you want to achieve. Your artists are going to resist. You had this nice list of 13 visemes and now you are creating more work. They don't know what an *incisivius labii* is and don't want to. You can explain that it is what makes Lara pucker up and they won't care. You will have to win the staff over by showing the creative possibilities for character expression that are now available. They probably still won't care, so get the producer to force them to do it. I have created a sample muscle set in Chart 1. This will give you some groups from which to pick.

Now I need to relate these individual muscle meshes to the viseme and emotional states. This is accomplished with "muscle macros" that blend the percentages of the basic muscles to form complex expressions. This flexibility permits speech and emotion in any language without the need for special meshes.

I still need to handle the case where several muscles interact with the same vertices. However, now there is a biological foundation to what you are doing

Certain muscles counteract the actions of other muscles. For example, the muscles needed to create the "oo" viseme (*incisivius labii*) will counter the effect of the jaw dropping (*digastric* for those of you playing along at home). One real-time animation package I have been working with called Geppetto, from Quantumworks, calls this Muscle Relations Channels. You can create a simple mathematical expression between the two to enforce this relationship. You can see this effect in Figure 5.

Now for the Animation

I finally have my system set up and my models created. It is time to create some real time animation. The time-tested animation production method is to take a track of audio dialog and go through it, matching the visemes in your model set to the dialog. Then, in second pass, go through it and add any emotional elements you want. This, as you can imagine, is pretty time consuming. Complicating the matter is that there are not many off the-shelf solutions to help you out. The job requires handling data in a very special way and most commercial animation packages are not up to the task without help.

Detecting the individual phonemes within an audio track is part of the puzzle that you can get help with. There is an excellent animation utility called Magpie Pro from Third Wish Software that simplifies this task. It can take an audio track and analyze it for

FIGURE 5. *W.C. Fields's jaw is open and then blended with the "oo" viseme. Image courtesy of Virtual Celebrities Productions and Quantumworks.*

The World's Premier Conference and Exhibition on Computer Graphics and Interactive Techniques

ENCOMPASSING:

3D modeling

animation

digital imaging

dynamical systems

gaming R&D

haptic displays

high-res technologies

rendering

simulation

virtual reality

visual effects

visualization

Web 3D

Images (left to right): inTouch © 1998, MIT Media Lab; Homage to Hilbert © 1998, Nelson Max, Lawrence Livermore National Laboratory; The Ceren Web Resource © 1998, University of Colorado at Denver; The Bush Soul © 1998, Rebecca Allen; Object-Oriented Displays © 1998, The University of Tokyo; The DNA Story © 1997, Andreas Koch, Digital Studio SA

+1.312.321.6830
siggraph99@siggraph.org

Conference 8-13 August 1999
Exhibition 10-12 August 1999

Los Angeles Convention Center
Los Angeles, California USA

SIGGRAPH 99

Sponsored by ACM SIGGRAPH

www.siggraph.org/s99/

phoneme patterns you provide automatically. While not entirely accurate, it will at least get you started. From there you can manually match up the visemes to the waveform until it looks right. The software also allows you to create additional channels for things such as emotions and eye movements. All this information can be exported as a text file containing the transition information. This in turn can be converted directly to a game-ready stream of data. You can see Magpie Pro in action in Figure 6

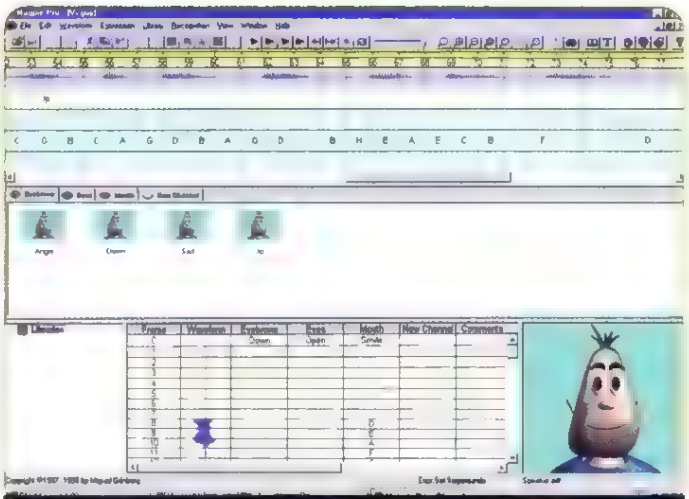


FIGURE 6. *Magpie Pro simplifies the task of isolating phoneme patterns in your audio track.*

Wire Me Up, Baby

With all the high-tech toys available these days, it may seem like a waste to spend all this time hand-synching dialog. What about this performance capture everyone has been talking about? There are many facial capture devices on the market. Some determine facial movements by looking at dots placed on the subject's face. Others use a video analysis method for determining facial position. For more detailed information on this aspect, have a look at Jake Rodgers's article "Animating Facial Expressions" (November 1998). The end result is a series of vectors that describe how certain points on the face move during a capture session. The number of points that can be captured varies based on the system used. However, typically you get from about eight to hundreds of sensor positions in either 2D or 3D. The data is commonly brought into an animation system like Softimage or Maya and the data points drive the deformation of a model Filmbox by Kaydara is designed specifically to aid in the process of capturing, cleaning up, and applying this form of data. Filmbox can also apply suppressive expressions, inverse kinematic constraints, and perform audio analysis similar to Magpie Pro.

This form of motion capture clearly can speed up the process of generating animation information. However, it's geared much more toward traditional animation and high-end performance animation. In this respect it doesn't really suit the real-time game developer's needs. It's possible to drive a real-time character by using the raw motion capture data to drive a facial deformation model. However, for a real-time game application, I do not believe this is currently feasible

In order to convert this stream of positional data into my limited real-time animation system, I would need to analyze the data and determine what visemes and emotions the performer is trying to convey. You need a filtering method that will take the multiple sample points and select the viseme or muscle action that is occurring. This is really the key to making motion capture data usable for real-time character animation. This area of research, termed gesture recognition, is pretty active right now. There is a lot of information out there for study. However, Quantumworks's Geppetto provides gesture recognition from motion capture data to drive "muscle macros" as both a standalone and a plug-in for Filmbox.

- Ekman, P. and W. Friesen. *Manual for the Facial Action Coding System.* Palo Alto, Calif.: Consulting Psychologist Press, 1977.
- Faigin, Gary. *The Artist's Complete Guide to Facial Expression.* New York: Watson-Guptill Publications, 1990.
- Goldfinger, Eliot. *Human Anatomy for Artists.* New York: Oxford University Press, 1991.
- Landreth, C. "Faces with Personality: Modeling Faces That Exude Personality When Animated." *Computer Graphics World* (February 1996): p. 58(3).
- Waters, Keith. "A Muscle Model for Animating Three-Dimensional Facial Expression," *SIGGRAPH* Vol. 21, N. 4 (July 1987): pp. 17-24.

Facial Animation
http://mambo.ucsc.edu/psl/fan.html

Gesture Recognition
http://www.cs.cmu.edu/~face

Performance Animation Society
http://www.pasociety.org

Magpie Pro
http://thirdwish.simplenet.com

Filmbox
http://www.kaydara.com

Geppetto
http://www.quantumworks.com

Where Do We Go from Here?

Between viseme-based and muscle-based facial animation, you can see that there are a lot of possible approaches and creative areas to explore. In fact, the whole field has really opened up to game development in terms of opportunities for game productions as well as tool developers. Games are going to need content to start filling up those new DVD drives and I think facial animation is a great way to take our productions to the next level. ■

Acknowledgements

Thanks to Steve Tice of Quantumworks Corporation for the skull model and the use of Geppetto as well as insight into muscle-based animation systems. The W. C. Fields image is courtesy of Virtual Celebrity Productions LLC (http www.virtualceleb.com) created using Geppetto. The female kiss image is courtesy of Tom Knight of Imagination Works (http://www.imaginationworksinc.com).

http://www.gdmag.com

MicroScribe-3D™

Desktop Digitizing Systems

MicroScribe 3D is an advanced 3D digitizing tool. Microscribe empowers you with the ability to capture complex 3D models from physical objects by tracing over their surfaces. The 3D digitizing process is fast, accurate, and more affordable than you might think!

The MicroScribe 3D is a desktop system that easily connects to your serial port and runs on PC, Mac, and SGI platforms. MicroScribe works directly in your favorite 3D graphics and CAD packages such as 3D Studio Max, Alias, Lightwave, AutoCAD, Solidworks, form Z, Rhino 3D, Amapi Studio, and many more. MicroScribe comes in several size and accuracy configurations to meet the demands of your project.

For details, call Immersion at 408-467-1900 or 1-800-893-1160

Immersion

Talking Heads: Hierarchical Facial Animation in Real-Time 3D

Until very recently, facial animation was a technique reserved for full motion video and prerendered cinematics. The continuing advances in rendering technology and geometry-dedicated processors have opened the door for animators to use this technique within the realm of real-time 3D entertainment.

This month's article is the first installment in a series dedicated to expanding the knowledge base of real-time facial animation. Over the next few months, we'll discuss facial anatomy and skinning techniques, as well as phoneme recognition and animation with linked expressions. We'll tackle the problem from the ground up, and by the end of the process we'll have covered all of the steps necessary to create and animate a speech-driven human head

Why Go to the Trouble?

There is absolutely no logical reason why the actors in today's real-time 3D games shouldn't be able to smile, scowl, and talk with the player. The struggle to create a believable virtual world is the struggle to create the illusion of reality. For any game based within a virtual environment, the player's enjoyment is directly linked to how immersive that environment feels. In the ideal case, players will forget that they are sitting in front of a computer screen, and will lose themselves for a few hours within the virtual worlds we create. It is incumbent upon us as developers to use every means at our disposal to generate this effect. Facial animation can and will be one of the most effective tools for achieving this. When in-game characters interact with the player through recognizable facial expressions and lip-synched spoken dialogue, we will have taken several steps towards achieving the perfect virtual world.

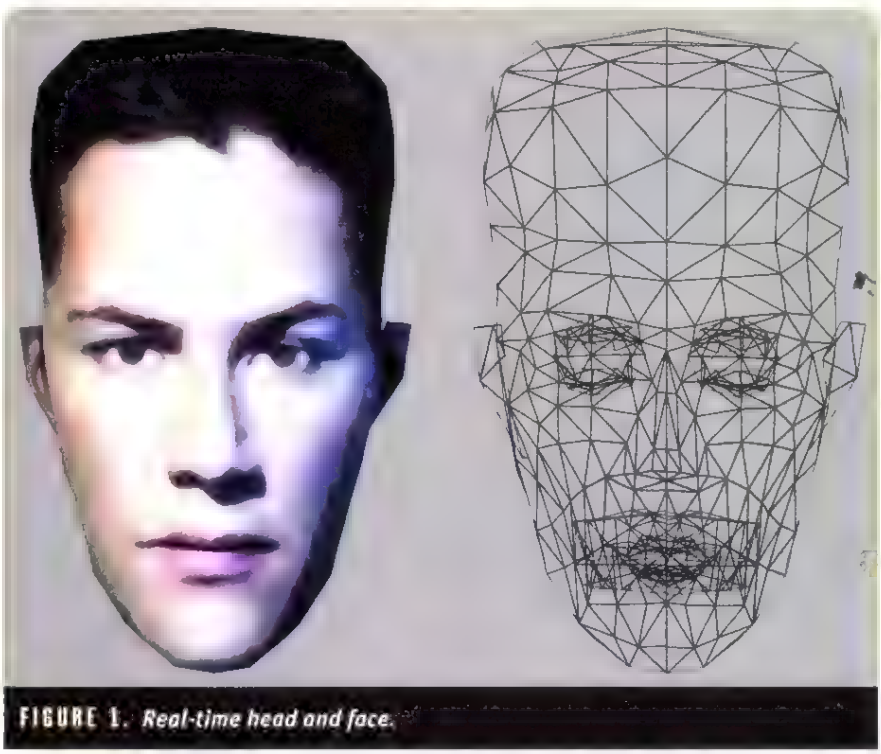


FIGURE 1. *Real-time head and face.*

Modeling the Head and Face

The main reason we haven't been able to create believable lip-synched characters has been the rendering engines' polygon limitation. In a human head, the skin of the face is very plastic, in other words, extremely malleable. In order to achieve this effect convincingly and without excessive distortion, the polygonal densities of the face are required to be fairly high. As a result of the recent improvements in rendering and pro-

Mel has worked in the games industry for several years, with past experience at Eidos and Zombie. Currently, he is working as the art lead on DRAKAN ( ). Mel can be reached via e-mail at mel@surreal.com.

cessing power, and through aggressive use of level-of-detail models, most cutting-edge engines will now accommodate the relatively high density meshes required for facial animation

result, setting up a skeletal hierarchy at the beginning can still get you there because the mesh that's produced for each skeletal animation can then be cloned and used as a morph

The main reason we haven't been able to create believable lip-synched characters has been the rendering engines' polygon limitation.

For our example we'll use a human head, although the basic principles discussed will apply to almost any face with the basic bilateral construction found in humans. The head in Figure 1 has been modeled with sufficient polygons for facial expression, but still low enough for current engine technology. Almost all of the head's 800 or so polygons have been devoted to the areas surrounding the eyes, nose, and mouth, where the bulk of facial expression is displayed. In this case, we've actually gone to the trouble to add the internals of the mouth, with tongue, teeth, and cheek surfaces included (this will be necessary if our character is going to be speaking close up, in an in-game cut-scene, for example).

Although this head has been modeled with real-life photographic reference, which is often the best resource, Hogarth's *Drawing the Human Head* (Watson-Guptill, 1989), and Fargin's *The Artist's Complete Guide to Facial Expression* (Watson-Guptill, 1990) are two excellent references with more generalized information.

Why a Skeletal Hierarchy Instead of Morph Targets?

The jury is still out on what is the most efficient method for creating animated facial expressions. While generating morph targets can sometimes be faster and often allows fine tuning of facial animation, taking the time to build a skeletal system with linked expressions can save time in the long run, especially when your game requires a large number of animations. In any case, if you're sold on using morph targets as your end target. The bottom line is that your technique will be determined by two factors: your engine's animation system, and your animator's expertise in the given method.

Expressive Regions of the Face

In order to set up our hierarchy correctly, we need to identify the areas of the head and face that will be animated. We get most of our information about a person's mood by looking at two distinct regions of the face. The highlighted areas in Figure 2 enclose the upper and lower "active regions" involved in facial expression. This is where we will invest the bulk of our time and energy setting up the hierarchy. The upper facial node acts as the parent to the nodes controlling the eyes, eyelids, and eyebrows. Similarly, the lower facial node is the parent for the nodes of the lower jaw, mouth, and tongue. Breaking down the hierarchy in this way will do two things for us. First, by compartmentalizing the areas of facial expression, we have broken down the problem of facial animation into smaller, more easily managed tasks. Second, by creating a preset list of animations for each region, we will be able to generate a widely varied set of facial expressions with relatively little effort.

Bone Structure and Facial Muscles

In order to create the motion of the human face accurately, a basic understanding of the underlying muscle and bone structure is needed. Figure 3 shows where each node corresponds to the mesh, while Figure 4 shows each node's relative position in the hierarchy. As you can see, with a few exceptions, the nodes are bilateral, and lend themselves to being moved in pairs.

1. The skull. This will serve as the top node for the hierarchy. Several of

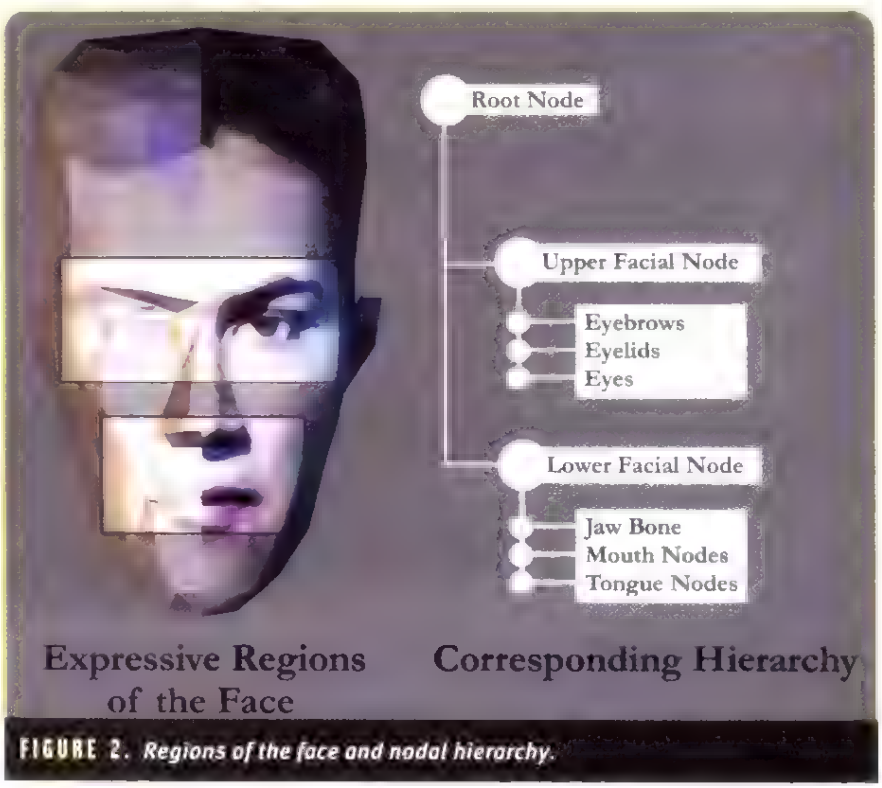


FIGURE 2. *Regions of the face and nodal hierarchy.*

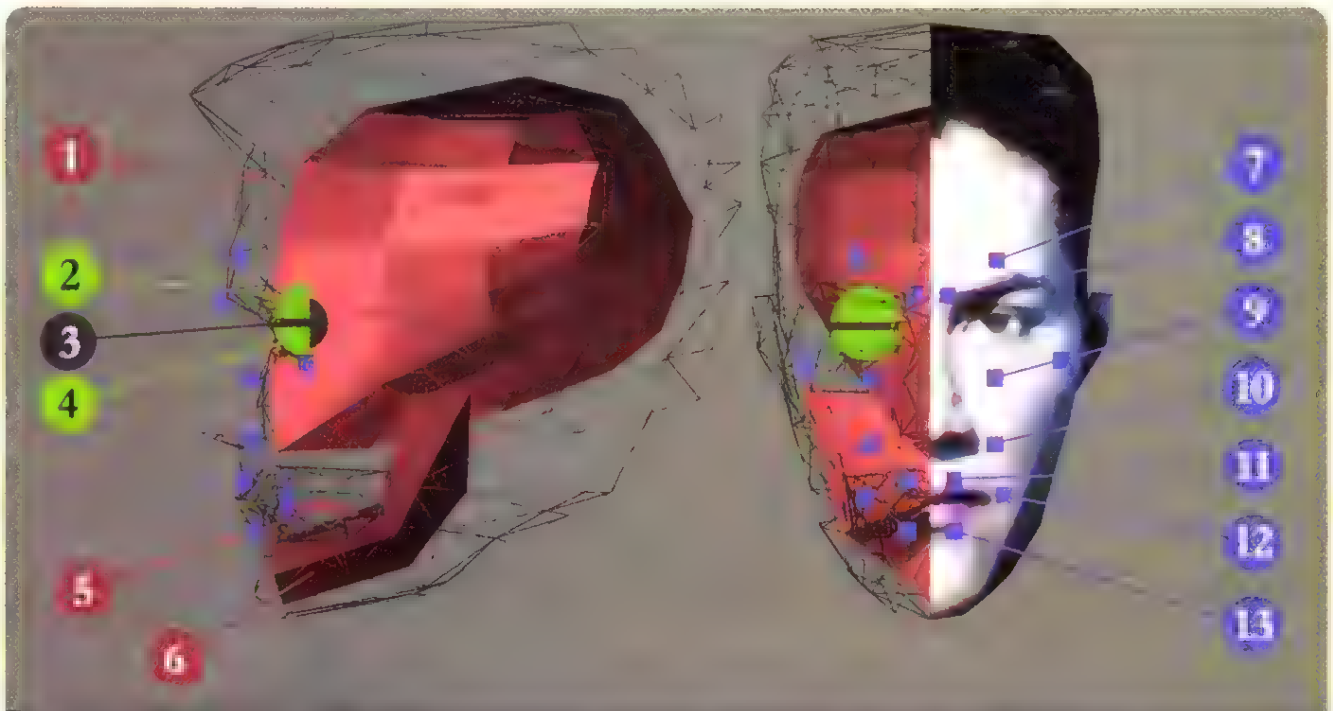


FIGURE 3. *Nodal placement.*

the muscle groups involved in facial animation are anchored to this bone. Although this node does not animate and could be represented by any arbitrary shape, it is useful to approximate the general shape of the skull when creating a hierarchy, as this serves as the foundation for and aids in the placement of subsequent nodes.

2. Upper eyelid node (*levator palpebrae*). This raises the upper eyelid, as in a surprise or fear response. Because the upper eyelid has its own muscle group and the lower does not, most of the motion associated with the eyelids occurs in the upper eyelid

3. Eye node. The mesh of the eyeball should be separated from the rest of the face so that it can turn freely with the node Also consider placing a constraining expression on the eye nodes, so that they move in concert

4. Lower eyelid node. The lower eyelid is kept open largely by the force of gravity. This node works in conjunction with number 9 below

5. Tongue nodes. The tongue is an optional part of the hierarchy, although if you want to use close-up shots of the characters, it is a definite necessity The muscles of the tongue are particularly versatile, and are among the strongest in the body. For this reason, the full flexibility of this muscle should be represented by no fewer than three bones or nodes.

6. The jawbone. Pay particular attention to the pivot point for this node. The jawbone is a hinged joint, with the pivot point located at the extreme rear point on the bone, where it is hinged to the skull. The tongue nodes and most of the lower mouth nodes are children of this node

7. The frontalis. This muscle is responsible for raising the eyebrows vertically. A bilateral sheet-like muscle, it is connected to the thick fiber of the scalp immediately beneath the hairline, and inserts into the skin directly under the eyebrows. The action of the frontalis is seen in such expressions as surprise, sadness, and fear, although it also sees action during regular conversation since raising the eyebrow is one of the most common facial gestures. We often do this in concert with or in place of hand gestures during normal speech. Although this muscle can be represented by a single node, many people have control over the individual sections of the frontalis, enabling them to raise one eyebrow, while lowering the other (the "Spock" eyebrow) For this reason, there are two nodes, one for each side of the face.

8. The corrugator. This muscle group, also known as "the scowling muscle," is actually comprised of two muscles, the corrugator and the procerus. These muscles anchor to the skull at the top of the nasal cavity, and at the inside corners of either eye socket The basic function of the corrugator is to pull the eyebrows down , while at the same time bringing them closer together. Used primarily in conjunction with the frontalis, this muscle is

For any game based within a virtual environment, the player's enjoyment is directly linked to how immersive that environment feels.

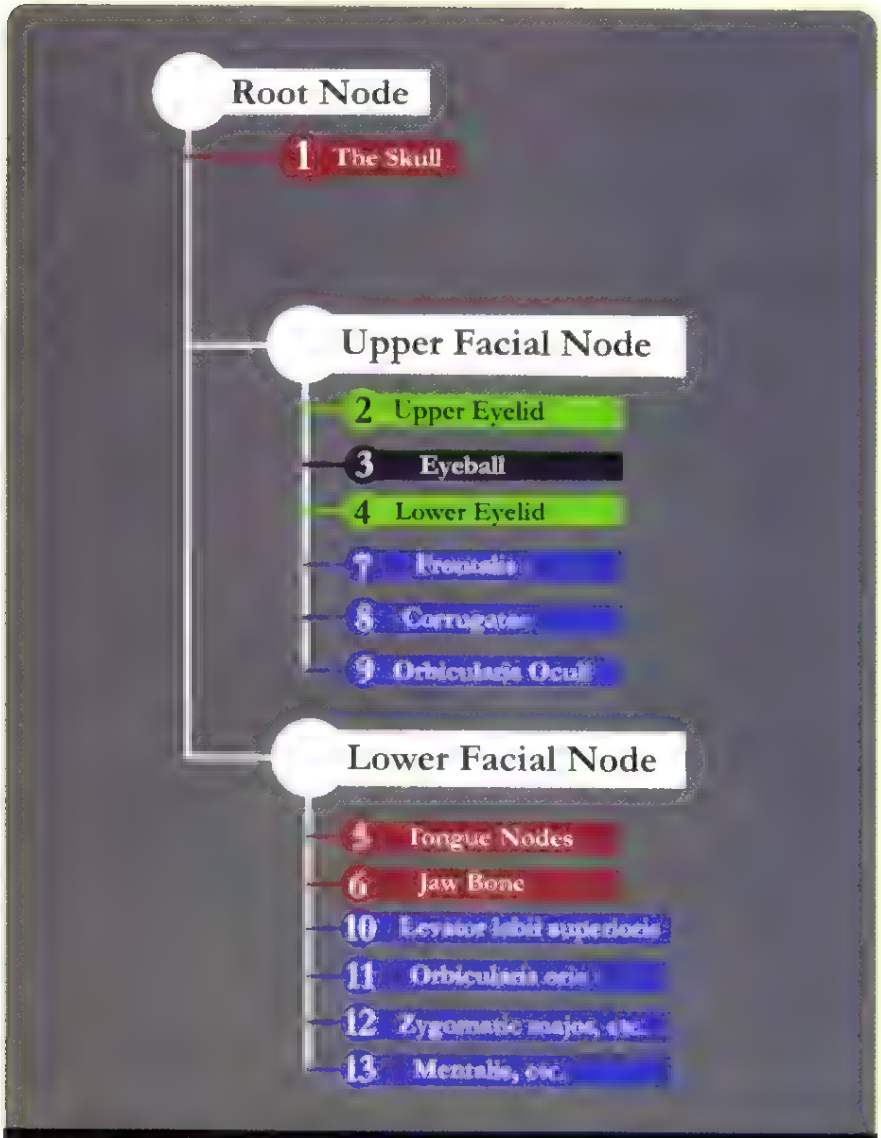


FIGURE 4. *Each node's relative position in the hierarchy.*

associated with the acts of crying, scowling, and extreme concentration.

9. *ORBICULARIS OCULI.* This is a large muscle group made up of concentric rings of muscle tissue, totally encircling the eye and extending into the cheek. As this muscle contracts, it tends to squeeze the eyes shut while at the same raising a good portion of the skin of the cheek. This muscle sees action whenever we squint, laugh, or smile, and is the primary operator in expressing pain Although this muscle group can be approximated by two separate nodes (as in Figure 3), a single spherical node encompassing the eye can also be used In that case, scaling the node down in the *x*, *y*, and *z* axis would approximate the contracting of the muscle group

10. *LEVATOR LABII SUPERIORIS.* Also called the "sneering muscle," this muscle group uses a three-point anchorage spanning from the bottom edge of the eye socket to the lower ridge of the cheekbone. The muscles converge to a single point and insert into the skin just above the upper lip. Contracting this muscle tends to raise the upper lip towards the nostrils. In the real world, this muscle seldom sees action, except when expressing disgust, disdain, or loathing. We approximate it here by using a single node, which for us will be additionally useful in properly shaping the skin of the cheeks over several different expression groups.

11. *ORBICULARIS ORIS.* This is a banded group of muscles just under the surface of the upper lip, which is anchored at each end by the muscles at the corners of the mouth. Also called the "lip tightener," this muscle works in conjunction with the *levator labii superioris* to create the expressions of disdain and loathing. It is also used for pursing the lips, as when someone experiences deep thought or concentration

12. *ZYGOMATICUS MAJOR, RISORIUS/ PLATYSMA,* AND THE *TRIANGULARIS.* These three muscle groups are responsible for pulling the corners of the mouth up, out, and down, respectively Approximated here by the action of a single node, these muscle groups are active over a range of expressions, from the exclamation of joy and pleasure, to the extreme stress of pain or a tragic loss. To effectively mimic the action of these muscles, it is necessary to incorporate most of the nodes in the lower region of the face. Test this out by smiling broadly or frowning severely, and you will see just how much real estate this muscle affects This will also be one of the most active nodes, since for any speech or mouth movement, this node will be used to approximate the motion at the corners of the mouth.

13. *DEPRESSOR LABII INFERIORIS* AND THE *MENTALIS.* The combined action of these two muscles tends to pull the lower lip down (as during speech), or to push it upwards (giving a pouting expression). Here again, the opposing motion of two separate muscle groups can be approximated by a single node.

Wrap Up

Now that we've identified the major facial muscle groups and created our hierarchy, it's time to apply the skeletal structure to the mesh, and weight the facial vertices appropriately. The last step will be to set up a lip-synching table and expression list, and from there we'll be able to start animating. That's where we'll pick up next month.

For more information about facial animation from a programmer's perspective, please review Jeff Lander's columns, "Read My Lips: Facial Animation Techniques" (Graphic Content, June 1999) and "Flex Your Facial Animation Muscles," which appears in this month's issue of *Game Developer.* ■

Technology Update: Dominatrix for Softimage

In May's column, "See Jane Walk," we examined character animation techniques used in the industry, focusing particular attention on motion capture. Until very recently, motion capture animation in Softimage was a painfully rigorous exercise that limited animators to using a skeletal hierarchy based on a nodal constraint system. In layman's terms, the skeleton which the motion capture data fit onto was not necessarily the same type of skeleton which animators preferred to use while animating. The mo-cap skeleton had to match the data exactly, or the animation wouldn't work. Subsequent data manipulation on a mo-cap skeleton often proved restrictive, so that mixing modes of classical animation with mo-cap was problematic at best. Thanks to the team at House Of Moves, mo-cap in Softimage just got a whole lot easier.

Taylor Wilson, the CTO of the Los Angeles-based motion capture studio, has spearheaded a technique dubbed Dominatrix, which works to separate the motion data from the skeletal hierarchy; in essence, this technique frees up the animator to use whatever style of skeleton with which he or she feels most comfortable. For example, say you want to use motion capture data for your character, but you've already started animating using another method. With the previous restrictions in Softimage (and most other animation tools), you would have to scrap all your previous work in favor of the mo-cap skeleton which matched the mo-cap data. Now, with Dominatrix, you can combine previously recorded motion capture data with other forms of character animation without ever having to change your skeleton.

How Does It Work?

The process is extremely simple. You merely provide House Of Moves with your character's skeleton (submitted in a Softimage .HRC file), and they do the rest. The motion capture data, originally stored in the Acclaim format, is washed through a set of proprietary tools which correct for any differences in proportion, orientation, and number of nodes between the standard Acclaim skeleton and your character's skeleton. Then the data is mapped onto your character's skeleton, and saved out as a Softimage .ANI file. Dominatrix can also generate IK constraint information to drive any of the limbs of your character. (This is in place of or in addition to using tradition positional/rotational keys at each joint.) The level of customization possible allows you to set up your character pretty much any way you want, with the simple restriction that the character's skeletal hierarchy has the same parenting relationships as the mo-cap skeleton. This is illustrated in Figure 1, which shows several humanoid skeletons all sharing the same hierarchical relationship (two arms, two legs, a torso, and so on). Note that the limbs of each skeleton are different in length (some have long arms, some short) and that the number of nodes in any given limb can differ between characters (the War Giant character, for example, has four nodes in his leg whereas the human character has only three). Dominatrix can accommodate all of these skeletons with the same set of motion capture data.

Since this is something I had to see for myself, I opted for a test run of Dominatrix using DRAKAN's main character, Rynn. Her Softimage skeleton is chock full of extra nodes in her chest, arms, and legs, and doesn't match up very closely to the Acclaim skeleton. Within a few days of sending off the Softimage skeleton to House of Moves, I received several martial arts animation files (see Figure 2) which had been mapped onto the character. When I imported these onto Rynn's skeleton they worked flawlessly, and I have to say it was the least painful experience with motion capture I've ever had. The technique is called Dominatrix, and it's available (at present, exclusively) from House Of Moves (http://www.moves.com).

—Mel Guymon

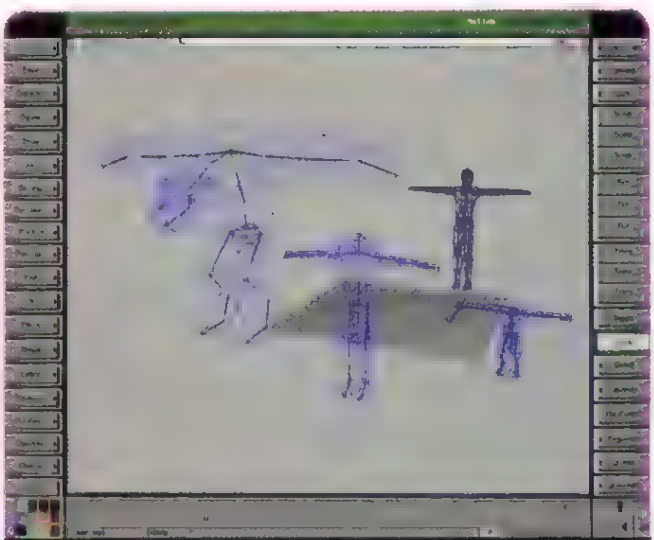


FIGURE 1. *Multiple Softimage skeletons.*

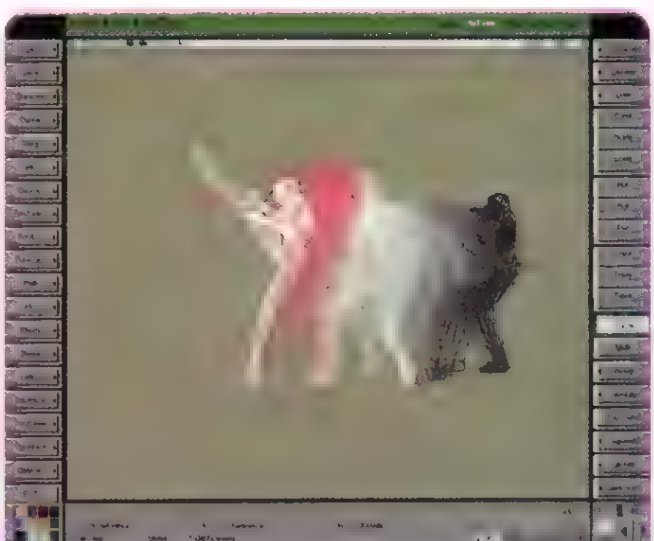


FIGURE 2. *DRAKAN's main character performing a martial arts move.*

My game is DOA if I don't ship on time.

CodeWarrior

CodeWarrior solves problems. Reduce your build time by as much as 90% with our industrial-strength software development tools. CodeWarrior tools are easy to use and they support multiple platforms, so you save even more time - shorten the learning curve, cut support costs, reduce porting times. CodeWarrior decreases production time and increases profit. No problem with that, right?

PROBLEM SOLVED CODEWARRIOR

1.800.377.5416

Find out how CodeWarrior can help you meet your deadlines:

www.metrowerks.com/deadlines

Mpath, HearMe and Mplayer

I remember talking to an Electronic Arts executive at Intel's Pentium III launch this past March. He was both roused by his company's plans in the online gaming market, and very self-conscious. I wondered if he was nervous about the prospects, or whether he was concerned that no one else would jump in and take a big bite of the pie. In April, when Mpath decided to go public, I had a chance to think some more about the issues facing the server side of the game business. There are some sobering lessons to be learned from Mpath, not only about online gaming, but also about what it takes — beyond games — to make it in cyberspace. If you look closely at Mpath's strategy you'll find that it could almost form the foundation of a big publisher, such as Electronic Arts, going wider than its core demographic, perhaps into the realms occupied by the really big fish, such as Disney Online.

The Online Gaming Business Model

Mpath was founded in January 1995. The company states as its business practice that it operates and licenses live community Internet sites to such companies as SegaSoft Networks. Mplayer.com is the company's low-latency gaming platform, or live community as it is called, that they deployed with the help of PSINet in late 1995. In January 1999 the company launched HearMe.com, its second live community site. While most game developers are familiar with Mplayer.com, they may not know as much about HearMe.com. HearMe.com currently consists of seven live audio communities, making live audio interaction available to people whose interests extend beyond entertainment. Mpath managed to raise approximately $34.9 million between its inception and the end of 1998 through the sale of equity securities to CSK, SegaSoft's parent company, and various venture capital (VC) and strategic partners. In January 1999 the company raised an additional $20 million through VCs to prepare for going public. That's a big chunk of change, but Mpath hoped to raise over $68 million from its initial public offering (IPO).

Being an online contender requires a big investment, even in the gaming niche that Mpath has carved for itself, but their long term strategy seems to indicate much more. The company derives its revenues from two business units, Live Communities and Mpath Foundation. Live Communities generate advertising revenues, and this includes advertisements targeting the key demographics at Mplayer.com and HearMe.com. Mplayer.com consumers are predominantly male. In sports and game player communities, more than 90 percent of the participants are male, and the people within those communities are typically between the ages of 13 and 50. However, approximately 40 percent of the participants in classic games and the casino community are women. The people within these communities are typically between the ages of 25 and 50. As for HearMe.com, currently more

FIGURE 1. *Mpath's statement of operations data for the fiscal years indicated as a percentage of total revenues. n/m means "not material."*

	1996	1997	1998
Net Revenues: Live Communities	9.7%	25.2%	37.6%
Foundation	90.3	74.8	62.4
Total revenues	100	100	100
Cost of net revenues:			
Live Communities	28.2	66.3	27.7
Foundation	72.6	22.7	9.8
Total cost of revenues	100.8	89	37.5
Gross profit (loss)	(0.8)	11	62.5
Operating expenses:			
Research and development	n/m	89.3	39
Sales and marketing	n/m	253.2	97.8
General and administrative	n/m	104.2	40.8
Stock compensation	n/m	61.5	32.4
Write-off of acquired intangibles	n/m	—	—
Total operating expenses	**n/m**	**508.2**	**210**

Omid Rahmat is the proprietor of Doodah Marketing, a digital media consulting firm. He also publishes research and market analysis notes on his web site at

than 40 percent of the participants are women, and its members are typically between the ages of 13 and 55

Mpath Foundation is the licensing and services arm of the company. In 1998, CSK Sega Sony and Electronic Arts accounted for 23, 12 and 10 per cent of total revenues respectively for Mpath Foundation. In 1997, CSK Sega and Sony accounted for 35 and 11 percent of total revenues respectively Combine both Live Communities and Mpath Foundation, and the business model for Mpath is simple The company created Mplayer.com, and now HearMe com, to showcase the application of its technology, and to put up a barrier to market entry for other parties interested in competing with so-called live communities There's a growing mesh of third parties that tie in with Mpath's live communities, whether they be advertisers, vendors who have products that sell directly through Mpath operations, or magazines and portals that link to Mpath's content graphics. Mpath Foundation then takes the technologies that make all this happen, and applies them to other places on the web. POP.X is the product that Mpath Foundation sells, a toolkit for enabling live communities And here is where it gets interesting and where the lessons emerge

Mpath is, like most Internet companies, primarily interested in growth This means pushing up memberships and subscriptions. In order to do that the company needs content from third parties, and that makes for more entertaining reasons to build an audience Furthermore, Mpath is also putting a lot of money into research and development of key technologies that will better manage its communities, and streamline the low-latency web entertainment experience

Growth Maxims

Mpath Foundation consists of a growing list of online entertainment companies, including CSK Sega, Electronic Arts, Fujitsu, GTECH, and I G Internet. The Mplayer.com service now comprises three active communities built around common interests and offers more than 100 of the most popular online multiplayer games According to Mpath's internal company reports, total usage time on Mplayer com exceeds 200 million user minutes per month as of January 1999, compared to 67 million user-minutes per month back in January 1998 Mplayer.com has become the tenth largest Internet site in terms of total usage time per month, according to the company's own usage data

The opportunities for Mpath are extensive. The technology the fact that they are growing beyond online gaming (a feat that some of their more esoteric competitors have failed to negotiate), and the general branding of the company's products all have helped the company achieve a leadership role. However, with almost two-thirds of their revenue coming from their technology licensing and services and advertising and subscriptions being very speculative means of making a profit on the Internet, Mpath faces some challenges

First of all there is the [illegible] franchise model It's worked for

THE ANTI-PIRACY AND E-COMMERCE SOLUTION FOR GAMES

TryNow™ [illegible] TryNow™ [illegible]

No source code modification is required.

TryNow™ demos can be distributed on CD ROM [illegible]

[illegible], TryNow™ protects games against [illegal] [illegible] equipping them with E-Tracer™ our patent pending, state-of-the-art anti-piracy technology based on software watermarking

No more 'dead end' demos With TryNow™ the demo itself can realize the sale

Protect your game and increase your sales potential now!

TryNow™ equips your game with the following:

- State-of-the-art anti-piracy protection
- Try & Buy functionality
- Full e-commerce enablement
- Level technology (Sell individual or grouped levels)

www.trynow.com
info@trynow.com
800-360-7650

trumedia™

Electronic Arts, and it means that publishers are willing to risk some effort and cash in order to create their own live communities. Being game developers at heart means that these publishers will probably want to build the server technologies themselves, and tailor them to specific gaming experiences and genres. In other words, the one-size-fits-all approach may not be the ideal way to go forward. Of course, Mpath also makes a good acquisition target for a company that wants to get a head start, and despite a very successful IPO, the volatility of Internet stocks could have an impact on the future growth of the company.

FIGURE 2: *Statement of operations (in thousands of dollars) from January 9, 1995, to year ending December 31, 1998.*

	1995	1996	1997	1998
Net revenues: Live Communities	$ -	$12	$686	$3,022
Foundation	-	112	2,041	5,005
Total revenue		124	2,727	8,027
Cost of net revenues:				
Live Communities	-	35	1,808	2,221
Foundation		90	620	790
Total cost of revenues	—	125	2,428	3,011
Gross profit (loss)	—	(1)	299	5,016
Operating expenses:				
Research and development	1,502	5,261	2,436	3,132
Sales and marketing	171	3,937	6,906	7,847
General and administrative	746	2,877	2,841	3,274
Stock based compensation	34	383	1,676	2,601
Write-off of acquired intangibles	—	12,876		
Total operating expenses.	2,453	25,334	13,859	16,854
Loss from operations	(2,453)	(25,335)	(13,560)	(11,838)
Interest and other income (expense), net	99	291	(93)	(111)
Loss before provision for income taxes	(2,354)	(25,044)	(13,653)	(11,949)
Provision for income taxes	(1)	(1)	(1)	(2)
NET INCOME	($2,355)	($25,045)	($13,654)	($11,951)

The real lesson to be learned from Mpath is that in order to succeed in the online world you have to juggle both your investment in sales and marketing to draw traffic and your spending on research and development to put up technology barriers to competition. These technology barriers are becoming increasingly difficult to maintain, relying as they do on similar objectives having to do with managing and maintaining online communities. Everyone wants to trap the consumer on their site and keep them there. As for spending lots of money to get traffic, that depends on content. Otherwise, your customer has to do very little in order to go somewhere else. Point, click and carriage return is about all it takes. And that's the best news that smaller developers have heard in a long time. Building communities around content, developing a franchise, and creating multiple revenue streams are all within the reach of any Internet savvy development group. As higher production costs raise the barriers to entry in the retail game market, and a handful of publishers control distribution, the online world offers the biggest means of access to the game enthusiast. The trick is keeping the consumer from turning the page, but isn't that the problem for every creative venture? Keep the audience coming back for more. Now go figure out how. ■

3DS MAX? Softimage? Lightwave? CAD?
PolyTrans Translates All Major 3D Formats!

PolyTrans v2
With 3DS MAX Plugin Support!

Release 2.2 Now Supports Native 3DS MAX Plugins, OpenFlight, Trimmed NURBS, Full IGES 5.3, Unique Animation Conversion (Lightwave /3DS/DirectX/Renderman) ACIS 4.2 SAT (solids) and more. Creates 'Render Ready' Files

New!

Download Demos From http://www.okino.com

3D Studio r4 & MAX
Apple 3DMF
Alias triangle
Detail
DirectX
DXF
IGES
Imagine
Lightwave
OpenFlight
PO

ProE slp
Renderware
Softimage hrc dsc
Sold Edge Part
Strata SPro v1.73
USGS DEM

PolyTrans is "the" premiere 3D conversion tool available toda

PolyTrans is by far the most versatile translator with the most reliable results for Modern Sales & Marketing Director, REM Infographics (3D Modern Boom)

PolyTrans is the only tool we have found that can correctly convert our 3D databases

The NuGraf Rendering System
Or, step up to Okino's NuGraf for full materia

Okino Computer Graphics
Toll Free 1-888-3D-OKINO, WEB: http://www.okino.com
T (905) 672-9328 F 672-2706 Email: sales@okino.com

3D Programming Talent

We're the award-winning creators of NASCAR '98 and '99 from EA Sports, Madden Football, Star Trek: Deep Space 9, Byzantine (Discovery Channel Multimedia), Starfire, Soccer Challenge (Purple Moon), La Russa Baseball, etc. We've earned a wide range of awards and nominations from the Codies, Emma's etc., as well as creating titles that make the console and PC best-seller lists. We also have a major new action game for PSX and N64 that was unveiled at E3, as well as a big-license RPG that will be announced later this year.

Founded in 1988 and now a company of 70 veteran gamers, we've worked hard to establish ourselves as one of the most respected game development houses in the business, delivering A+ titles to top publishers including Electronic Arts, Discovery Channel Multimedia, Maxis, Sony, SSI, Purple Moon and Amer.can Online.

Over the past 10 years, Stormfront has consistently demonstrated steady moderate growth, good stability and a strong personnel retention record. We are now considering several new project opportunities for which we would like to add staff.

We're looking for a Development Director, Technical Director, Senior or Lead PSX Programmer, Senior or Lead PC Programmer for racing, 3rd person action adventure, a Mario 64–Crash Bandicoot style game, and a kids' title.

We're looking for exceedingly talented people who can play an integral role in creating our next generation of blockbuster game titles.

If you're looking recognition for individual contribution and an exceptional salary, stock option (we're privately held) and benefit package, please send your resume to:

Stormfront Studios, Attn: M. Daglow, 4040 Civic Center Drive
San Rafael, CA 94903 Fax: 415-461-3865, e-mail: mdaglow@aol.com
www.stormfrontstudios.com

Experience the be-all, end-all online resource
for the game development community...

gamasutra.com

The Art and Science of Making Games

GAME Developers CONFERENCE

Game DEVELOPER

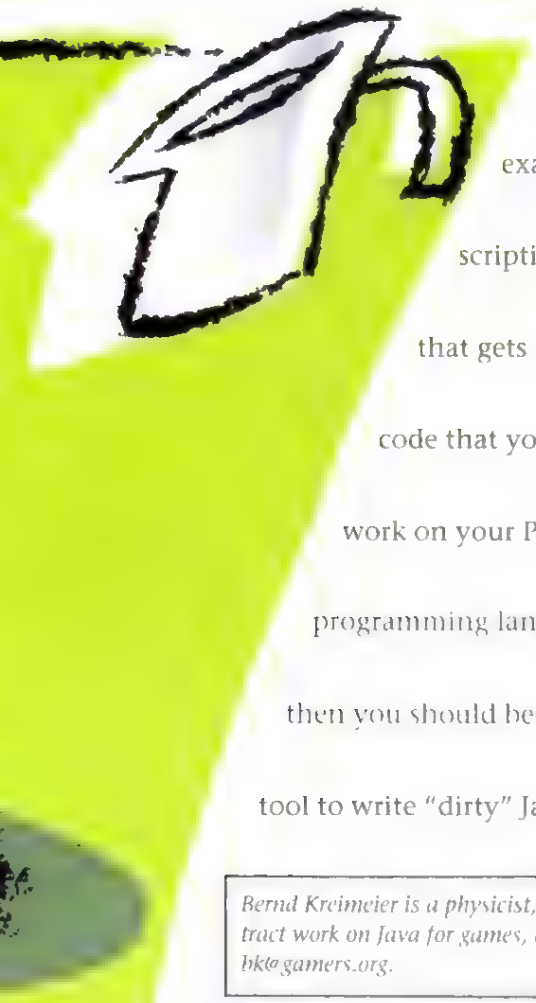
Illustration by Jackie Urbanovic

Dirty Java

Using the Java Native Interface within Games

BY BERND KREIMEIER

You have heard of Java. Actually, you probably have had a hard time trying to escape the hype surrounding it since 1995. There are, however, compelling reasons to use Java in shrink-wrapped games. Two major options that have recently been examined by professional game developers are using Java as your scripting language, and using Java for safe run-time downloadable code that gets executed on the client. In both cases, it is the interface to native code that you end up dealing with the same glue that is required to make Java work on your PC in the first place. If using a portable, standard, object-oriented programming language with built-in security within your game sounds appealing, then you should become acquainted with the Java Native Interface (JNI), which is your tool to write "dirty" Java — Java code that is tightly integrated with your native code.



Bernd Kreimeier is a physicist, a writer of novels and articles, and a coder. Currently living in Ireland, he is doing contract work on Java for games, and working on Warped Space, his own game design. This gun for hire — contact bk@gamers.org.

There is an abundance of information about "pure" Java, and this is not the place to explain all the actual and alleged advantages of Java technology. Unfortunately, many of the highly touted "pure" solutions quietly omit the sophisticated native machinery at work under the hood. Instead of looking at game applets running in web browsers, this article sizes up possible real-world uses for Java, and looks at the ways some game developers are already using Java for their titles.

A Brief Recap of Virtual Machines

Briefly, here are the key components that are relevant to this discussion of Java:

JAVA BYTECODE. This is pseudocode for a stack-based processor described in the Java specification. Valid bytecode has to satisfy a lot of requirements, but if you really wanted to, you could actually write it by hand with a decent hex editor and the specifications.

THE JAVA VIRTUAL MACHINE (JVM). Java CPUs have not yet conquered the market, so software must translate Java instructions into the language the PC hardware can understand. The JVM is a multithreaded program for your operating system, supplied from various vendors, that executes Java bytecode and maps bytecode to native instructions. The Java specification places few restrictions on the actual implementation of a JVM. You can find Open Source JVMs on the Internet, or even write your own clean-room implementation. You can also get the sources of Sun's reference Java Development Kit (JDK) and negotiate a license for commercial use.

THE JAVA PROGRAMMING LANGUAGE. This is an object-oriented language, designed with a subset of C++ in mind. It has run-time bounds checking, it is restricted in terms of memory access and handling, and it has simplified inheritance patterns. The language features an overwhelming inflation of extension APIs and core classes.

It is important to understand that these three components are entirely separate. A JVM will happily execute valid bytecode that was not generated from Java source — for instance, from a compiler that maps a language like Scheme to bytecode. You could also implement a simplified virtual machine that skips validation and executes bytecode no compliant JVM would accept, or create a virtual machine that has no garbage collector thread at all. In addition, there are compilers that generate native code from Java source files — you don't need a JVM to write an application in Java. From the "amusing avenue of hand-tuned virtual assembly" (as John Carmack of id Software referred to virtual CPUs) to using a JVM without ever coding in Java (converters mapping C or C++ subsets to Java are feasible), all your tampering needs can somehow be addressed.

A lot of confusion originates from dis-

FIGURE 1. *JNI built-in data types.*

Java	JNI Name	Values/Size	JNI Signature
boolean	jboolean	JNI_TRUE / JNI_FALSE	Z
byte	jbyte	signed 8 bits	B
short	jsnort	signed 16 bits	S
int	jint	signed 32 bits	I
long	jlong	signed 64 bits	J
float	jfloat	IEEE 754 32 bits	F
double	jdouble	IEEE 754 64 bits	D
char	jchar	Unicode* 16 bits	C
void	void	N/A	V

* The char data type is the only unsigned integral type available in Java. It is interpreted as 16bit Unicode, and mapped from/to UTF-8 encoding in I/O translations, but it is expanded to int for arithmetic operations, and all integral arithmetic operations are available.

FIGURE 2. *JNI visible class tree.*

Java Class name	JNI handle	JNI signature
java.lang.Object	jobject	Ljava/lang/Object
java.lang.Class	jclass	Ljava/lang/Class
java.lang.String	jstring	Ljava/lang/String
java.lang.Throwable	jthrowable	Ljava/lang/Throwable
N/A *	jarray	N/A *
java.lang.Object[]	jobjectArray	[Ljava/lang/Object
java.lang.String[]	jobjectArray	[Ljava/lang/String †
boolean[]	jbooleanArray	[Z
char[]	jcharArray	[C
byte[]	jbyteArray	[B
short[]	jshortArray	[S
int[]	jintArray	[I
long[]	jlongArray	[I
float[]	jfloatArray	[F
double[]	jdoubleArray	[D

* There is no java.lang.Array class, so JNI's jarray has no real equivalent in Java, just like jmethodID, jfieldID or jvalue.

† There is no jstringArray, so arrays of Strings map to the jobjectArray handle. So do arrays of Class, Throwable and any other subclass of Object, as well as mixed arrays and arrays of arrays.

cussions that fail to distinguish the various components of Java technology. All of these are required parts of any JDK however. The only JDK that is actually called JDK is the reference implementation provided by Sun Microsystems for Solaris and Win32, and its licensed ports to other platforms

The concept of a virtual machine for game scripting is not new, and not restricted to Java. John Carmack recently decided to use a custom virtual machine and bytecode created by a modified C compiler (LCC by Fraser and Hanson) for id's upcoming game, QUAKE 3: ARENA. Carmack once pointed out that game coders "have more urgent things to do than design languages." Ironically, he is now engaged in designing his own virtual machine and native interface. Technologies like Java's just-in-time compilation and HotSpot optimization originate in the Java technology mainstream, and they are powered by more resources than a single game company could ever command. If you can make Java work for your game, then you will benefit from this momentum

Talking to the Natives

Game coders usually do not trust cross-platform APIs based on layers of abstraction. Interpreted bytecode does not typically appeal to an industry that still counts on assembly to get a performance edge. Windows-based games sell, period, and portability is not really an issue.

Compare this to the holy grail of "100 percent pure Java." Mainstream Java technology is seemingly meant for tiny "gamelets," not serious games. Besides, despite the bloat, there are bits and pieces missing from the Java core classes: access to certain devices and system services is simply not available (and might never be, for design and security reasons), and politics sometimes gets in the way (witness the lack of Java OpenGL bindings). However, if you take a closer look, it turns out that there is always native code at the very heart of all that "pure Java": there is a JVM written in native code, and core classes partly implemented as native code. Here reigns the Java Native Interface, gluing together Java and native C/C++ code, and it is the key to combining your native code with Java technology

JNI is part of the Java specification and it's a mandatory part of all Java implementations. Sun was recently granted a court injunction forcing Microsoft to add JNI to the Microsoft Java implementation. Ideally, .DLLs and binaries using JNI should be bytecode-compatible for a given platform. The 1997 JNI specification is available online, and there are also books on the subject, so this article includes only a brief summary of it before we get into its applicability in games.

The first task JNI must solve is getting the JVM and user-written native code to agree on built-in types and memory layout to exchange data (see Figure 1). Only some of the core classes are represented for the native code (see Figure 2, most of them arrays of the built-in types.

What about `jclass` and `jobject`? JNI will not hand you the memory layout of a Java object, but it must provide you a handle. It even preserves the relationship between `java.lang.Object` and `java.lang.Class`. A `jclass` object can be cast to `jobject` safely in any JNI that complies to the Java specification (non-compliant implementations have been found). JNI is foremost aimed at C (the C++ bindings are just inlined wrappers), and no support for object-oriented programming on the native side is offered. With the exception of `Throwable`, `String`, and arrays, all classes have to be squeezed through the `jobject` and `jclass` representation. Arrays of arbitrary classes (including `String`) will always be mapped to `jobjectArray`. JNI defines `JNI_FALSE`, `JNI_TRUE`, a `jvalue` union type, and a `jsize` for your convenience

Further, handles have access to fields and also call methods of classes and objects. It might look like `java.lang.reflect.Field` and `java.lang.reflect.Method` are the Java equivalents to JNI's `jfieldID` and `jmethodID`, but JNI predated the Reflection API, and actual reflection support was added only to the latest JNI revision

Caching field and method IDs is a good idea, as retrieval involves a string

LISTING 1. *Native code making use of Java.*

```
// Java class, server-side Game Logic scripting.
package somegame;
class ScriptEngine {
  // This uses native error() callback
  public static void init() {..}

  public static native void error( int code );
}
#include <jni.h>
// Native error callback provided to Java
extern void SV_ErrorScriptEngine( jint i );
JNINativeMethod svSE = {"Java_somegame_ScriptEngine_error","(I)V", SV_ErrorScriptEngine };
// Native function to set up scripting module
void Sv_InitScriptEngine( JNIEnv* env ) {
  jclass clazz = NULL;
  jmethodID method = NULL;
  jint err;
  // Lookup class, loads the class if not yet done
  clazz = (*env)->FindClass( env, "somegame.ScriptEngine" );
  if ( clazz != NULL ) {
    // Register native method, returns zero on success
    err = RegisterNatives( env, clazz, &svSE, 1 );
    // Lookup Java methodID
    method = (*env)->GetStaticMethodID(env, clazz, "init", "()V");
  }
  // Handle errors

  // Call static method
  (*env)->CallStaticVoidMethod(env,clazz,method);
}
```

lookup. Be warned that caching can get tricky in applications with multiple threads and class loaders. You will have to keep an eye on the garbage collector as well — without a strong reference acquired by `NewGlobalRef()`, the garbage collector might remove the object your native code is still referring to. Likewise, dangling references not removed by `DeleteGlobalRef()` can keep obsolete Java objects from being collected. Use `DeleteLocalRef()` to avoid accumulating temporary references within loops. JDK 1.2 offers limited support for weak references, too.

Within your native code, all will revolve around the `JNIEnv` interface function table — your door to the Java side. It provides methods to handle references, create objects, load classes, access fields and call methods. Further, you get utility functions to iterate arrays, throw exceptions, and perform monitoring to make the native code threadsafe. Finally, an executable can also register functions as native methods, making code known to the JVM without dynamic linkage.

Listing 1 is a small example showing how native code can use Java to get things done, and how a native callback is registered with the JVM. Listing 3, discussed later on, does the opposite: it shows how Java calls native code.

It is tempting to use static (class) methods, as you do not have to handle an object in addition to the class. In many cases this is absolutely sufficient — servers will likely not run two script engines in parallel. In many other cases though, this leads to bad object-oriented design on the Java side.

LISTING 2. *Passing Arguments from Java to C.*

```
// Minimal Java control code wraps legacy engine.
package somegame;
class GameMain {
  protected static native int nativeMain( String[] args );
  public static void main( String[] args ) {
    int ret = nativeMain( args );
    ..
  }
static { System.loadLibrary("Game"); }
}
// Handing over the arguments to native code.
// This code will be put into the Game.DLL.
#include <jni.h>
extern int gameMain( int argc, char** argv );
JNIEXPORT jint JNICALL
Java_somegame_GameMain_nativeMain ( JNIEnv* env, jclass cls, jobjectArray jargv ) {
  jint res;
  jint argc;
  jint i;
  jboolean isCopy,
  jstring jstr;
  jsize len;
  const char* cstr;
  jargc = env->GetArrayLength(jargv);
  for ( i = 0; i<argc; i++ ) {
    jstr = (jstring)(*env)->GetObjectArrayElement(env,jargv,i);
    cstr = (const char*)((*env)->GetStringUTFChars( env, jstr, &isCopy ));
    // We copy - we have to release, and we don't want to accumulate local references.
    argv[i] = (char*)malloc( strlen(cstr)+1 );
    strcpy( argv[i], (const char*)cstr );
    // Did the JVM copy as well?
    if ( isCopy == JNI_TRUE ) {
      (*env)->ReleaseStringUTFChars( env, jstr, cstr );
    }
    // Clear local reference.
    (*env)->DeleteLocalRef(env,jstr);
  }
  // Call our main() now
  res = (jint)gameMain( (int)argc, argv );
  // Release allocated memory
  for ( i=0; i<argc; i++ ) {
    free( argv[i] );
  }
  // Return to Java.
  return res;
}
```

Double Indirection: The Catch-22

There have been competing native interface APIs proposed, most notably Microsoft's Raw Native Interface (RNI). The problem with RNI is that it exposes the underlying operating system and JVM implementation, which makes it impossible to port to another VM.

In some ways, the problem with JNI is that it does not expose the VM implementation. JNI makes you go through pains to ensure that native code never gets to see how Java objects are laid out in memory. Consequently, the native code has to deal with indirections every step along the way, many of them ultimately leading to table and string lookups. This is not good for application performance. See the example in Listing 2, where we pass command line arguments from Java to C, which involves references, arrays, and conversion to UTF-8 (the canonical two-byte Unicode encoding used by Java).

Tools such as `javah` generate C header files containing proper function prototypes (name and signatures) from a Java class. These tools are already Unicode-aware, thus using underscores and other special characters in Java method and class names can lead to surprising results. The code in GAME.DLL will be linked to the class by the JVM by calling `java.lang System.loadLibrary("Game")` automatically.

The example in Listing 2 implements a minimal Java wrapper around native

legacy code. Given all the implicit and explicit copying, we somehow seem to have come full circle: to get rid off some portability-related Java overhead, we decided to use native code, only to find out that the JNI design hampers the interaction between Java and native code to ensure portability. Now what? Well, there are basically two ways left to increase performance:

1. Brute force. You could switch tools and compile to native code. If you pursue this option, make sure your Java compiler supports JNI as well, and that it doesn't just compile pure bytecode.

2. Smart design. You could accept the limitations of the JNI and design your native and Java modules in a way that streamlines the interface between them.

Mind you, your native code by itself will be as fast as it gets. It's only the transfer of parameters and results back and forth that, inside an inner loop, incurs significant performance penalties.

The Invocation API

Now that you have seen some means to glue Java and native code together, where does a game developer actually get access to the virtual machine? The common answer is the Invocation API. The Invocation API allows you to embed the JVM into your native applications. It provides the means by which you can retrieve an existing JVM attached to your application, or launch one with proper configuration settings. Listing 3 shows how to invoke the JVM in an application, using JDK 1.2. You can use code like that shown in Listing 1 to get Java classes loaded and executed. If you do not want to encapsulate native method code into .DLLs, then your application can use the JNI function RegisterNatives() to make native functions from the executable known to the JVM. That way your game would ship without any .DLL.

Most JVM and compiler implementations fully support JNI, as it is needed to handle cleanly implemented core classes. (Interestingly, even Sun's own JDK does not always use JNI internally.) But the Invocation API was sometimes omitted from JDK ports and third-party JVMs. If you want to use the Invocation API, make sure your tools and targets support it.

Worse still, some JDK ports support Invocation, but do not do so properly (including some revisions of Sun's own Solaris reference implementation). Invocation requires threadsafe .DLL handling, which is not always granted (for instance, in some Solaris and Linux revisions). If the dynamic linking is not threadsafe, your application will suffer spurious errors during startup. Furthermore, the official Java specification now sanctions limitations of the reference JDK that affects DestroyJavaVM(). It is not possible to destroy a JDK JVM. Consequently, you can't invoke another one from the same application. Multiple JVMs, whether subsequent or in parallel, are not possible. Once you lose your JVM for whatever reason, your application must terminate. Fortunately, most of the other pitfalls were smoothed out last year.

In Listing 3, ignore the JVM handle which you can always retrieve by calling the JNI function GetJavaVM(). A more flexible approach to invoke the JVM calls GetCreatedJavaVMs() first to check whether a JVM already exists, and uses AttachCurrentThread() to make itself known if one is found. The question is, do you really have to invoke the JVM this way?

LISTING 3. *How to invoke the JVM in an application using JDK 1.2.*

```
// JDK 1.2 Invocation example
#include <jni.h>
// Setting some standard options
extern jint JNICALL Printf( FILE *f, const char *fmt, va_list args );
extern void JNICALL Exit( jint code )
extern void JNICALL Abort( void ),
#define NOPTIONS = 5,
JavaVMOption OPTIONS[5] =
{
  { 'classpath", (void*)"C:\\java\lib\classes.zip; D:\Game\classes" },
  { "verbose",   (void*)"jni,gc' },
  { "vfprintf",  (void*)Printf  },
  { "exit",      (void*)Exit    },
  { "abort",     (void*)Abort   }
},
/  Create a JDK 1.2 JavaVM as desired
JNIEnv* SV_InitJavaVM( JavaVMOption* options, jint nOptions ) {
  JavaVMInitArgs vm_args,
  JavaVM*    vm_handle; // not preserved
  JNIEnv*    env;    // return to caller
   int       ret,
   / Request version 1.2
  vm_args.version = JNI_VERSION_1_2;
  ret   JNI_GetDefaultJavaVMInitArgs(&vm_args);
  if , ret==0 ) {
    vm_args options = options
    vm_args nOptions = nOptions
    vm_args ignoreUnrecognized = JNI_TRUE,
    ret   JNI_CreateJavaVM  &vm_handle, &env, &vm_args );
    if ( ret==0 ) {
      return env,
    }
  }
        error handling
}
```

Two Architectures: Embedded Java and Encapsulated Native Code

How you choose to obtain the JVM for your game is, in all likelihood, the most important decision you have to make when rigging up a Java-based game project. To a C coder, invocation

might seem a natural choice. This architecture is known as embedded Java. Your application is linked to a .DLL that provides a JVM, which happily lives and dies within your application, completely at your disposal. It looks like this:

```
// Engine piggy backed with JVM or
// Engine retrofitted with JVM
    set options, possibly parsing commandline
nOpt = SV_GetOptions(&options,argc, argv);
    Invoke JVM, get script engine started
SV_StartVM( SV_InitJavaVM(options, nOpt));
// Start the actual game
return gameMain( argc, argv );
```

On the other hand, if you write a pure Java game, or use the somegame GameMain class shown in Listing 2, then some other application loads the JVM and hands it the Java bytecode of your game. This scenario is used when a web browser runs "gamelets" for example, or when JDK's java loads an executable JAR file. Whether your game uses native code or not, you do not have to concern yourself with invocation if the main loop is written in Java. Native method code will be encapsulated in Java classes, as long as the .DLLs required are loaded in time. It does not look like much of a difference, but choosing one or the other might have a huge impact on your project.

Embedded Java: A Natural Choice?

Let's look at an example that I call the "QUAKE 3 scenario." Your team has nearly finished a game engine written in C or C++. The game has a large and stable legacy code base that you don't want to tamper with, yet there is a clear-cut need that Java might address, such as a new server-side scripting language, or support for client-downloadable code. In short, you want to retrofit an existing application with a Java component.

The history of the QUAKE engine is a great example. QUAKE featured a custom scripting language (QuakeC), QUAKE 2 introduced a server-side .DLL (GAME.DLL), and some QUAKE engine offspring now deploy client-side .DLLs.

Embedding is possibly the best answer in all cases where you have to deal with C legacy code that is not implemented in an object-oriented fashion. The JVM is just another device that is initialized, configured, started and shut down again. There are some restrictions (for instance, you cannot restart the JVM once it has shut down), but in general, all you do is provide raw data (bytecode) to the embedded JVM much the same way you'd feed .WAV files to a sound device. If you do not want to use .DLLs at all, embedding is your solution. You also get a lot more control over the JVM that is used by your game. Shipping a Java Runtime Environment (JRE) with an embedded solution might save you support and maintenance headaches. It might also address some reverse engineering, tampering and cheating issues.

If embedded Java is used, either C control code executes Java methods on the JVM which return the data, or the Java code in turn calls native methods to write back. You could have Java threads run in parallel to your application, but debugging an application that moves back and forth between native and Java execution stack frames can be a challenge to you and your tools — multiple threads will make it even tougher.

What problems are specific to using an embedded JVM? Some have already been mentioned, such as negligence or outright omission of the Invocation API from some Java implementations, and potential problems you might face when falling back on compiling Java to native code. All of these problems can be overcome one way or the other, however. The real danger might be much more subtle.

Your legacy code has a certain design — possibly not object oriented at all if you used C, or possibly an object-oriented design that maps badly to Java if you used C++ excessively (if you made use of templates and/or multiple inheritance). In these cases, taking a single component of your game (for instance, the server-side game logic) and converting it to Java could introduce bugs and errors in formerly stable and tested code. Worse still, through JNI the design used in native code will proliferate into Java code, resulting in badly designed Java code. For example, if you never handle objects (see Listing 1 and its use of class methods), it is unlikely that you are using an object-oriented design. Legacy code tends to share memory using pointers for speed and convenience, which is not possible with JNI. You have to think hard and make judicious cuts to get a lean interface between Java and native code. High levels of abstractions

Java wraps native code in PRAX WAR

Written by Billy Zelsnack during his days at Rebel Boat Rocker, PRAX WAR was destined to be the first major game to use Java for most of its code. With the exception of a C renderer, the PRAX WAR engine was written entirely in Java. Zelsnack explained the game this way: "We use JDK and JNI. The game itself starts from Java. I use Java as controller code for C. Java is very good at calling C code, but [it is] not necessarily as clean the other way around." The design kept raw data (such as textures and sounds) on the native side, but made them accessible to Java as needed. Billy Zelsnack found few problems with the core classes (at one point, UDP networking performance was an issue), and found no problems with the most feared Java component, the garbage collector. The engine used just two threads — one thread that listened for incoming packets, plus the main loop itself.

Unfortunately, Rebel Boat Rocker's publisher, Electronic Arts, decided earlier this year to cancel the project, stating that the game had "missed its technology window". If Electronic Arts had had the kind of faith in Rebel Boat Rocker that Sierra has shown in Valve Software, we might have found that their assessment was straight to the point — maybe Prax War missed its technology window by being too *early*.

Zelsnack summed up his Java experience up this way: "Java opened up possibilities for the product that could not have been realized without its power. It was one of the things I was most excited about and proud of."

implemented as abstract base classes and interfaces usually work best: the more details you hide, the better JNI will work for you.

Consider handling structured data on both sides of JNI, such as that used for collision handling. Collision response is part of the game logic (does the player take damage, bounce, or die?) and is thus handled in the Java code in our example. Collision detection might be performed within the scene representation that is also used by the rendering code—almost surely native code you want to keep. In this scenario, your Java game logic might call native code to trace an object's movement through the scene.

This is where the level of abstraction is relevant. Take the QUAKE 2 representation of a vector in 3D space: `float[3]`. In Java, this is best represented as a class with `float x,y,z` fields. This avoids array bounds checking overhead, and frees us from worrying whether JNI pins or copies the array. For objects that small and likely to have all their fields accessed, the simplest way to pass them back and forth is to unfold them on the stack as primitive data types, much the same they would be flattened for serialization.

This solution is more the exception than the rule, however. In general, it pays off to hide as much detail as possible on both sides. The game logic does not need to know whether axis-aligned bounding boxes or spheres are used for collision detection, it only has to initiate updates to position and size. For the actual trace in native code, it is irrelevant whether a given entity within the bounding volume is a player, a monster, or a fireball.

Using a high level of abstraction on the Java side by sticking to abstract base classes and interfaces makes retrieving and caching method IDs in your native code much easier, since all objects within an inheritance tree will share the same signatures. You might find it safer to cache field and method IDs in class descriptor structs or C++ objects. Field access is more efficient, but exposes the internal implementation of your Java objects. JNI methods like `GetFieldID` and `GetFloatField` can be used instead of, say, `GetMethodID` and `CallFloatMethod` to access instance fields directly.

You pass references as `jobject` handles instead of pointers to make Java data accessible to native code. The reverse is not possible: neither C structs nor C++ objects are visible to Java. You can address C++ objects or C structs with `jint` handles on the Java side, using more (hash table look-up) or less (typecast) safe ways to retrieve the effective address. A proxy class would then wrap native methods with public accessors, like that sketched out in Listing 4.

Of course, if you want to avoid switch statements in the native method, or you want to wrap C++ accessor methods instead of exposing

LISTING 4. *Java Proxy for a C++ object.*

```
package jni;
class Proxy {
  /** Handle to retrieve C++ object, native side. */
  private final int handle;
  /** Native LUT/constructor. */
  private static final native int newNative();
  /** Constructor, gets/creates a handle to a native object. */
  public Proxy() {
    handle = newNative();
  }
  /** Simplified fieldID. */
  private final int SOME_FIELD = 1;
  /** Accessor hiding the simplified retrieval. */
  public final float getSomeField() {
    return getFloat( handle, SOME_FIELD );
  }
  /** Method that saves us many retrieval() functions. */
  private final native float getFloat( int handle, int field );
}
// Minimal C++ object, and JNI glue
class NativeObject {
  public: NativeObject( jobject owner ) {
    this.owner = owner;
  }
  public: jobject getOwner() {return owner;}
  public  float someField;
  public  jobject owner;
};
#include <jni.h>
extern jclass InvalidProxyOwnerException;
extern jclass InvalidFieldIndexException;
// extern C implied
JNIEXPORT jint JNICALL Java_jni_Proxy_getFloat
  (JNIEnv*  env,
   jobject  owner
{
 return (jint) new NativeObject(owner))
}
JNIEXPORT jfloat JNICALL Java_jni_Proxy_getFloat
  (JNIEnv* env,
   jobject owner,
   jint handle,
   jint field
{
  // Truly dirty. Trust on blank finals
  NativeObject* obj = (NativeObject*)handle;
    if ( obj->getOwner() != owner
      *env)->ThrowNew(env, InvalidProxyOwnerException, "access attempted by non-owner");
    switch( field ) {
    // Enums to be kept in sync manually
    case 1: return obj->someField
    default:
      *env)->ThrowNew( env, InvalidFieldIndexException,
            "access attempted to non existing field");
    }
  }
}
```

fields, you could also implement the public Java accessor as a native method Incidentally, maintaining the same set of enums in Java and native code is one of the problems that does not yet seem to have an elegant solution. If you are using a look-up table to retrieve pointers for handles, the jobject argument might already be sufficient.

A native proxy implemented as a C++ object or C struct could cache a global jobject reference along with method IDs and field IDs. Caching actual game data inside native code means that your proxies have to be kept synchronized with the master objects, or you will end up with consistency errors that are very difficult to track down.

Alternatively, you could encapsulate the results or take a snapshot of a native object's state in a new Java object created in native code, using the JNI function NewObject() to call a Java constructor. This approach works even better if your native and Java modules communicate by passing event descriptor objects to a queue

Encapsulated Native: A Magic Bullet?

If you go down the road of Java control code, be prepared to throw out legacy code whenever necessary Encapsulation means dividing and splitting your code base into tiny pieces — heaven if you are at liberty to design from scratch, hell if you have to handle code that is just sticking together. If you can't isolate native code modules and wrap them with Java classes, then there won't be a secure migration path Handling a native legacy code base might well take more time than gutting it and starting from scratch. If you consider abandoning C/C++ as your main language, an encapsulation architecture is definitely the way to go

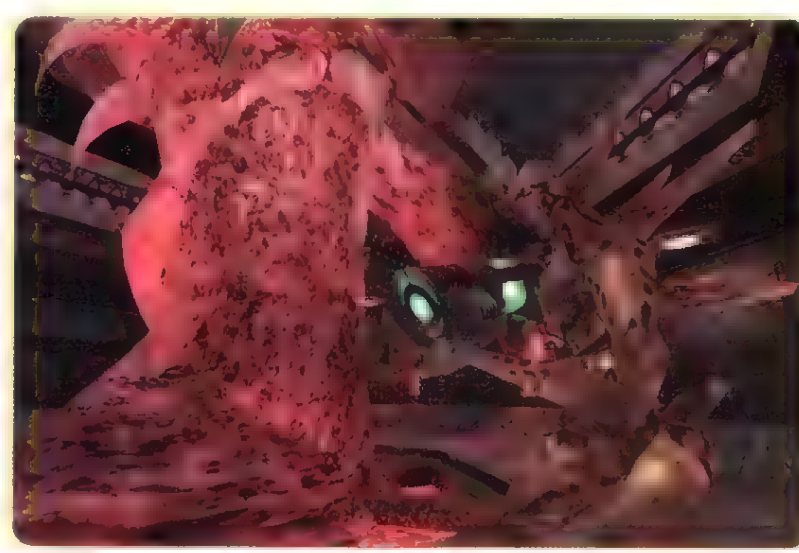
Java and JNI will always have some performance disadvantages that make them unsuitable for time-critical inner loops. However, it can be very efficient write your control code in Java (which can account for up to 90 percent of a game's total code base) even though that often takes up less than 10 percent of the overall processing time — especially for games that make the CPU spend half of the time in an OpenGL driver. This was the reasoning behind the PRAX WAR architecture that Billy Zelsnack implemented at the now defunct game development studio, Rebel Boat Rocker (see sidebar, "Java wraps native code in PRAX WAR"). There are only a few areas in which native code is really needed, such as managing raw data (textures and sound resources, for example), and rendering. Collision detection might best be done in native code shared with the renderer. Collision response, however, is a natural part of high-level game logic. In some cases, the lackluster performance

id Abandons Java for QUAKE 3: ARENA

John Carmack considered using Java in id's games for quite some time, ever since he announced that the company was leaning towards client-downloadable code for the Trinity project. "The QA game architecture so far has two separate binary .DLLs: one for the server-side game logic, and one for the client side presentation logic." Games that licensed the Quake 2 engine, notably HALF-LIFE and HERETIC 2, also wound up using client side .DLLs. However, with the hacking attacks on QUAKE 2 servers in mind, Carmack states that, "While it was easiest to begin development like that, there are two crucial problems with shipping the game that way: security and portability. If we were willing to wed ourselves completely to the Windows platform, we might have pushed ahead, but I want QUAKE 3: ARENA running on every platform that has hardware-accelerated OpenGL and an Internet connection."

His solution: "I had been working under the assumption that Java was the right way to go, but recently I reached a better conclusion. The programming language is interpreted ANSI C. The game will have an interpreter for a virtual RISC-like CPU." UNREAL followed a similar approach: companies that license the engine can opt to use compiled C or C++ code, and interpreted UnrealScript is available for homebrew scripting.

The advantages of using a C or C++ subset for your VM are obvious when it comes to handling legacy code. Ironically, it was Java portability problems that led id to develop the QUAKE 3 custom VM. Sun's promise of "write once, run anywhere" did not hold for the Invocation API on important server platforms, so Carmack decided to abandon the embedded JVM he had planned to use. "My ideal situation," he stated, "would be to include the interpreter in the QUAKE3.EXE, and just treat class files as data to be loaded and dealt with like anything else." Unfortunately, while this solution works fine on Win32 platforms, this was not guaranteed for Linux, OS/2, or even Solaris. "Having made the decision to do my own interpreter, I feel much more at ease not having to rely on anyone else's external code. When it comes around to the next development cycle, I will make the Java decision again." As for embedding: "We are still working with significant chunks of an existing code base. If I did want to go off and start fresh, I would likely try doing almost everything in Java."



of Java core classes might force you to replace them with your own custom Java code, or even use some native code instead. In the end, you will have a few cleanly separated native code modules you can optimize to your heart's content, controlled by robust Java code. The object-oriented design propagates top-down into your native code, which should be another benefit.

The Holy Grail: 100 Percent Pure Java

Supporting the multitude of Internet server platforms (Solaris, Linux and other UNIX flavors, OS/2, and Windows NT) has become increasingly important for multiplayer games. Presuming the existence of decent Java networking core classes and acceptable performance of Java-based scene lookup and collision detection code, a dedicated server implemented entirely in Java is an attractive possibility -- portable by default and in the absence of JNI, easily compiled to native code.

Portability issues are not as pressing for clients, the majority of which are Win32-based. At the same time, a real need for native code might only be found for the client, which is pushing a lot of raw data (textures and sounds) from local disk to local memory to native driver code. Unfortunately, only a few games, such as id's experimental QUAKEWORLD release, have separated the client and server completely. Consequently, a dedicated Java server means a separate code base that partly duplicates the shared client/server sources. Dedicated servers have become quite common recently, but in the long run, the code duplication is not acceptable. Automated Java-to-C or C-to-Java conversion might offer a temporary workaround only.

Ultimately, shipping a client written in Java will require decent Java bindings around reliable, cross-platform APIs, and these are nowhere to be found. Java does not have official OpenGL bindings, there is not even a portable native API for 3D sound, and the politics surrounding Sun's proprietary Java3D scene graph API doesn't help matters. For the foreseeable future, commercial games will not be feasible without JNI and native code.

Q2Java Today: What's Next?

Few games (Red Storm Entertainment's POLITIKA is among them) have shipped with Java built into them, but if you want to see a full-sized example of an embedded Java VM running QUAKE 2 deathmatch right now, you should visit the Q2JAVA web site at

Q2JAVA, orginally by Barry Pederson, is a cooperative open source implementation of the QUAKE 2 multiplayer game logic and works with the native QUAKE 2 executables on Windows 95/98/NT, Linux and (as a dedicated server) Solaris.

This article has introduced the two major roads to using Java for your game, though admittedly, a lot of details have been omitted and major issues (like security) were not touched upon.

However, Gamasutra.com is hosting more of my Java game development articles, including a web version of this article that includes an annotated list of references. ■

- See the JNI specification at:
- See Rob Huebner's 1999 GDC slides at:
- Gordon, Rob, et al. *Essential JNI: Java Native Interface*. Prentice Hall Computer Books, 1998.
- See the extended version of this article on Gamasutra.com, which includes further links and resources.

Embedded Java in VAMPIRE: THE MASQUERADE

In a recent developer update, Nihilistic's Director of Technology, Robert Huebner, stated: "I've always been rather anti-Java; all the Internet hype surrounding the language was overwhelming. But after examining the language further, it was clear that it makes an ideal scripting language for games. The embedded Java API allows us to provide our designers with a subset of the Java environment, and the JNI interface allows us to provide hooks from the Java Virtual Machine (JVM) directly into the game engine." The new Java-based system will replace a custom compiled language, COG, that the team used in its previous title, JEDI KNIGHT: DARK FORCES 2. According to Huebner, "The JVM is a lot faster than the systems we wrote ourselves; their kernel is more heavily optimized, the available Java compilers produce much more optimized object code, and the newest JVM systems include just-in-time (JIT) compilation to native instructions as a standard feature. And since the language is so much richer than our previous C-subset, it gives the designers a much wider range of possibilities." Because Nihilistic is developing primarily for Windows, it is able to apply solutions that were not feasible for id Software's multi-platform QUAKE 3: ARENA strategy.

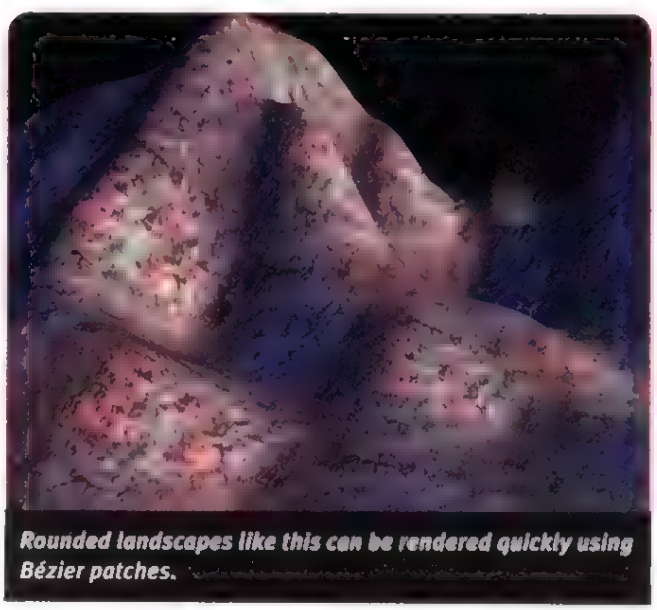
Optimizing Curved Surface Geometry

By Brian Sharp

In my article last month, I covered the basics of curved surfaces, including Hermite curves, Bézier curves, and Bézier surfaces. I warned that the implementations would be straightforward and naïve, and therefore slow.

That was last month. The purpose of this article is to take what we learned last month and delve into optimization strategies and algorithms so that our slow, straightforward code can become fast (but still straightforward) code.

This article ends with a more interesting demo than last month's — a terrain system of Bézier patches. Therefore, this article presents information that bridges last month's information and this month's terrain system. We'll find a better patch tessellator, quickly talk about terrain light map generation and why it's a good idea, and finally talk about putting patches together to form the terrain and the bevy of complications that accompany it.



Rounded landscapes like this can be rendered quickly using Bézier patches.

Central Difference Tessellation

At the end of the last article, I created a demo of a single Bézier patch, tessellated uniformly, bright red, that spun around at a woeful frame rate. Certainly, this was nothing you could use to build a robust application. The computations involved for just that one patch included 32 cubic function evaluations, 32 quadratic function evaluations, two vector normalizations, a vector cross product, and OpenGL lighting for each point. We can do two things to speed that up: do less work per point, and, even more importantly, calculate fewer points.

Perhaps the worst sin of my naive implementations was the patch tessellator, `uniformPatchTessellator`. It just calculated a ten-by-ten grid of points on the patch. It calculated each point explicitly and just about as slowly as it could. If we can devise a better patch tessellator, we're a long way towards a truly useful implementation.

There are better tessellation algorithms out there, and I'll discuss one particular algorithm that I've used to make these Bézier patches really crank: central differencing. One form of the central differencing algorithm is mentioned in Watt and Watt's *Advanced Animation and Rendering Techniques*.

When Brian's not coding up a storm or sleeping through meetings at CogniToy, he's filling up otherwise useful pages of the magazine with his nonsensical ramblings. Keep him busy by writing to him at brian_sharp@cognitoy.com with questions or comments or else he might find the time to write again.

(Addison-Wesley, 1992), but this version is modified significantly from the one they present.

The fundamental mathematical construct used in central differencing is the Taylor polynomial. A Taylor polynomial of a function is a polynomial that approximates that function near a certain point. So, let's say you've got a sine curve, and you start atop a peak of the curve. Then, you can form polynomials that look like a sine curve near that peak. As the polynomial becomes more and more complex, it looks more and more like that sine curve. The formula for the Taylor polynomial of a function $h(x)$ near a parameter u is

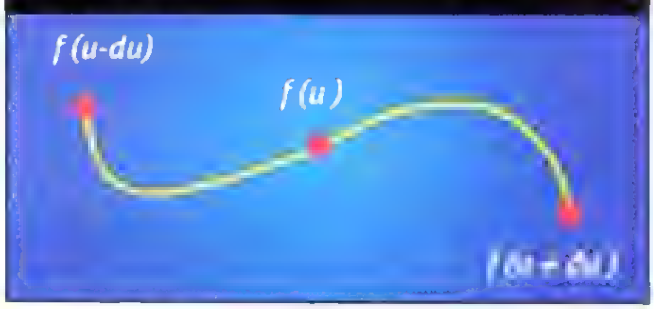
$$h(u+du) = \sum_{i} \frac{(du)^i}{i!} h^i(u) \qquad \text{where } h^i \text{ is the } i \text{ derivative of } h.$$

Then, in practice, you pick some limit other than infinity for the summation, and you get an approximation of the function — the higher the limit, the better the approximation. However, in our case, we're dealing with cubic functions. Therefore, any derivative after the third will be zero, so it's pointless to make the upper bound anything higher than three. So, if our cubic function is $h(x)$, its Taylor polynomial will look like this:

$$h(u+du) = h(u) + du(h'(u)) + \frac{du^2}{2} + h''(u) + \frac{du}{4} h'''(u) \qquad \text{Eq. 1}$$

You've probably noticed by now that all this does is take one cubic polynomial and give us another cubic polynomial. You're probably wondering what that does for us. This is where central differencing takes over. The idea behind central differencing is that, given information about the endpoints of a curve segment, we'd like to be able to find the midpoint. That way, we can then recurse on both sides of the midpoint and find their midpoints, and so on, until we have our curve.

FIGURE 1. *The parameters used in the Taylor polynomials for central differencing.*



Rather than keep you hanging, I'll mention the process right now and go through the derivation of it afterwards. If we have two points along the function $h(x)$, say $h(a)$ and $h(b)$, and we want to find the point halfway between $h(a)$ and $h(b)$, or $h((a+b)/2)$, we do the following. First, we average $h(a)$ and $h(b)$, which gives us just a point on the line between them.

Next, we take $h''(a)$ and $h''(b)$, the second derivative of h at a and b. We average them to find the second derivative at the midpoint, $h''((a+b)/2)$. We also find du, which is half the parameter distance between $h(a)$ and $h(b)$, or $(b-a)/2$. Then, we compute $-du^2 * h''((a+b)/2) / 4$. We add that to the averaged point we found above, and that's our point. The pseudo-code for this process is shown in Listing 1.

LISTING 1. *This code takes two points on a cubic curve and the second derivatives of the curve at those points and finds the midpoint and its second derivative.*

```
CentralDifference(h(a), h''(a), h(b), h''(b))
{
  // Get the midpoint parameter value.
  mid = (a+b)/2;

  // Get the midpoint of the line between
  // h(a) and h(b)
  avgPoint = (h(a) + h(b)) / 2;

  // Find the second derivative of
  // h((a+b)/2).  Since the second
  // derivative is linear, we can
  // just average them.
  h''(mid) = (h''(a) + h''(b)) / 2;

  // Find du, the distance from h(a)
  // to h((a+b)/2).
  du = (b-a) / 2;

  // Now h(mid) is this next term
  // plus avgPoint
  secondTerm = -(du*du)*h''(mid)/4;
  h(mid) = avgPoint + secondTerm;

  return h(mid) and h''(mid)
}
```

Central Differencing: The "Why"

So we've established that central differencing is a better way to tessellate a patch. Now we need to choose some values so that the Taylor polynomial makes this work. We'll call the midpoint $h(u)$, and du is the distance from the midpoint to an endpoint. That makes the first endpoint $h(u-du)$, and the other endpoint $h(u+du)$; Figure 1 shows an example curve. From that, we can evaluate the Taylor polynomial as though we were solving for the endpoints, given the midpoint. That gives us the following two equations:

$$h(u+du) = h(u) + du(h'(u)) + \frac{du^2}{2} h''(u) + \frac{du}{4} h'''(u)$$

Eq. 2

$$h(u-du) = h(u) + -du(h'(u)) + \frac{(-du)^2}{2} h''(u) + \frac{(-du)}{4} h'''(u)$$

Eq. 3

In Equation 3 we can pull the negative sign out of the $-du$ terms on the right side, giving use these two equations:

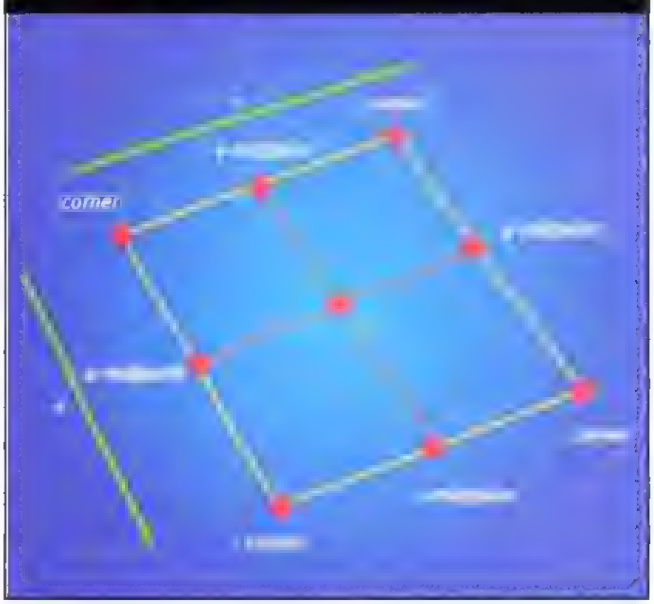
$$h(u+du) = h(u) + du(h'(u)) + \frac{du^2}{2} h''(u) + \frac{du}{4} h'''(u)$$

Eq. 4

$$h(u-du) = h(u) + -du(h'(u)) + \frac{du}{2} h''(u) + \frac{-du}{4} h'''(u)$$

Eq. 5

FIGURE 2. *One level of recursion in central difference patch tessellation.*



Equations 4 and 5 by themselves do not give us enough information to solve for the midpoint $h(u)$, but we can reduce them to a single nice equation just by adding them together.

$$h(u + du) + h(u - du) = 2h(u) + du^2h''(u) \qquad \text{Eq. 6}$$

The positive du and negative du terms cancel out, which is why we have such a nice equation. However, we don't want to solve for the endpoints — we need the midpoint. Rearranging the equation we get:

$$2h(u) = h(u + du) + h(u - du) - du^2h''(u) \qquad \text{Eq. 7}$$

Dividing through by 2 and breaking things up, it becomes:

$$h(u) = \frac{h(u + du) + h(u - du)}{2} - \frac{du^2h''(u)}{2} \qquad \text{Eq. 8}$$

That first big fraction on the right side of the equation is just the average of the two endpoints. That's easy enough to find because we're assuming we have whatever information we need about the endpoints. The second fraction on the right side is a little harder, though. It depends on the second derivative at the midpoint. Can we get that from the endpoint? Well, note that the second derivative is just a linear equation. That's easy enough to find from the endpoints; we can just average the second derivatives of h at the endpoints, and we've got it:

$$h''(u) = \frac{h''(u + du) + h''(u - du)}{2} \qquad \text{Eq. 9}$$

Now, throwing that back into equation 8, we've got:

$$h(u) = \frac{h(u + du) + h(u - du)}{2} - \frac{du^2(h''(u + du) + h''(u - du))}{4} \qquad \text{Eq. 10}$$

That's a mouthful, but that's it. What that says is that we can find the midpoint by averaging the endpoints and adding on a weighted average of the endpoints' second derivatives. Since we know we can find the midpoint's second derivative (we just did it in Equation 9) we'll have enough information when we're done to recurse and find the rest of the curve.

That's just for a curve, though, as h is only a function of one variable. We want do this for patches, which are functions of two variables. But before we move on, we should take note of an interesting property of the central difference.

Nonlinearity

In Equation 10, we find the midpoint by averaging the endpoints and adding on the other term, the weighted second-derivative term. What's interesting is that if that second term is 0, then it means that the midpoint lies on the line between the endpoints. Therefore, we'll refer to that second term as the *nonlinear term* of the central difference — it determines how far the midpoint is from the line between the endpoints.

As you might suspect, this is a great heuristic that we can use to decide how far to tessellate a curve. We just set a certain threshold, and when the magnitude of the nonlinear term is below that threshold, we stop recursing and return. Therefore, we'll have much more detail in very curved areas, and not nearly as much detail in the less curved areas. This is a perfect way to save detail for the parts of the curve that need it. We'll use the nonlinear term later on when we put together heuristics for our patch tessellation.

Central Differencing Revisited

We've derived enough to tessellate a curve with central differences. Now we need to figure out how to tessellate a patch with central differences. As it turns out, central differences in two dimensions are slightly harder than the one-dimensional variety. So, we describe our problem again. This time, we want to be able to take information about four corner points on a patch and find the same information about the midpoints between the corners and also the center point. An illustration of this is Figure 2.

Looking at the illustration, it's clear that if we can find the midpoints of this one level given the corners, we can recurse within each of the four smaller squares using the same method, and so on, until we have our patch. Now, we aren't sure exactly what information we need at the corner points, so let's see what it would take to find the other five points.

The u-midpoints can be generated using the one-dimensional central differencing. We can ignore v, since it stays constant, and use the endpoints and their second derivatives with respect to u to find the u-midpoints and their second derivatives with respect to u.

The v-midpoints can be generated the same way. We can ignore u and use the endpoints and their second derivatives with respect to v to find the v-midpoints and their second derivatives with respect to v.

To keep this all straight, it's useful to keep track of what information we have for which points:

$$corners \rightarrow f(u,v), \frac{\partial^2 f(u,v)}{\partial u^2}, \frac{\partial^2 f(u,v)}{\partial v^2}$$

$$u-mids \rightarrow f(u,v), \frac{\partial^2 f(u,v)}{\partial u^2}$$

$$v-mids \rightarrow f(u,v), \frac{\partial^2 f(u,v)}{\partial v^2}$$

$$center \rightarrow nothing$$

Before we worry about finding the center point, we should make sure the *u*-midpoints and the *v*-midpoints are complete. That is, if we want to recurse, we need to make sure that we end up with the same exact information at the new points as we got from the corner points. Otherwise, that information won't be available to the four smaller squares when we try to recurse. So, the objective is to find the second partial derivative of *f* with respect to *v* at the *u*-midpoints, and the second partial derivatives in *u* at the *v*-midpoints. However, it's not clear quite how we do this, since the one-dimensional central differencing doesn't say anything about how to interpolate a variable of *v* along *u* or vice-versa. A clue comes from the equation of the Bezier patch:

$$\sum\sum p_{ij} B_i^3(u) B_j^3(v)$$

Let's find the second partial in *v* along *u* first. We must find the second derivative of the patch with respect to *v*. From the above equation, we'd have:

$$\sum\sum p_{ij} B_i^3(u) \frac{d^2 B_j^3(v)}{dv^2}$$

Now, the basis function with respect to *u*, *B(u)*, is a cubic. Since it's untouched, we can see that the second derivative of the patch with respect to *v* is a cubic function in *u*. If we have any cubic in *u*, we can interpolate it in the *u* direction using the one-dimensional central differencing.

In this case, then, the function we're interpolating is the second derivative with respect to *v*:

$$g(u) = \frac{\partial^2 f(u,v)}{\partial v^2}$$

So, we can interpolate *g(u)* if we have *g(u)* at the endpoints and *g*'s second derivative with respect to *u* at the endpoints. That means that to get the second partial derivative in *v* at the *u*-midpoints, the corner points need this data:

$$\frac{\partial^2 f(u,v)}{\partial v^2}, \frac{\partial^4 f(u,v)}{\partial u^2 \partial v^2}$$

The corners already have the first value, the second derivative with respect to *v*. We have to add the second one, the second partial derivative with respect to *u* of the second partial with respect to *v*.

Now we've got the second partials in *v* at the *u*-midpoints. We can easily find the second partials in *u* at the *v* midpoints in the same way if we just swap the variables. So, the second partial in *u* is a cubic function in *v*, so we can find it by adding the following to the corner points:

$$\frac{\partial^2 f(u,v)}{\partial u^2}, \frac{\partial^4 f(u,v)}{\partial v^2 \partial u^2}$$

Again, the corners already have the first item. Also, we can switch the partials around, so

$$\frac{\partial^4 f(u,v)}{\partial v^2 \partial u^2} = \frac{\partial^4 f(u,v)}{\partial u^2 \partial v^2} \quad \text{Eq. 11}$$

Finally, we note that this term is a linear function of *u* and *v*, so we can get it at the *u*-midpoints and the *v*-midpoints just by averaging the values from the corner points.

Therefore, we have all the information we need in the corners, and we can get that information back at the *u*-midpoints and *v*-midpoints. Just to keep track, here's what we're up to:

$$corners \rightarrow f(u,v), \frac{\partial^2 f(u,v)}{\partial u^2}, \frac{\partial^2 f(u,v)}{\partial v^2}, \frac{\partial^4 f(u,v)}{\partial u^2 \partial v^2}$$

$$u-mids \rightarrow f(u,v), \frac{\partial^2 f(u,v)}{\partial u^2}, \frac{\partial^2 f(u,v)}{\partial v^2}, \frac{\partial^4 f(u,v)}{\partial u^2 \partial v^2}$$

$$v-mids \rightarrow f(u,v), \frac{\partial^2 f(u,v)}{\partial u^2}, \frac{\partial^2 f(u,v)}{\partial v^2}, \frac{\partial^4 f(u,v)}{\partial u^2 \partial v^2}$$

$$center \rightarrow nothing$$

Now, we just have to fill in the center point, and we're done. (Don't worry, we're almost there.)

Referring back to Figure 2, the center point can be derived from the *u*-midpoints just as each of the *v*-midpoints was derived from its two corner points. So we use the *u*-midpoints and their second partials in *v* to find the center point and its second partial in *v*. Then, we can get the center's second partial in *u* by averaging the *v*-midpoints' second partials in *u*. Finally, we can get the center point's final value, the second partial in *v* of the second partial in *u* (the value from Equation 11) by averaging the values from the *v*-midpoints.

We're done! It took a while, but we finally have all the information for all the points:

$$corners \rightarrow f(u,v), \frac{\partial^2 f(u,v)}{\partial u^2}, \frac{\partial^2 f(u,v)}{\partial v^2}, \frac{\partial^4 f(u,v)}{\partial u^2 \partial v^2}$$

$$u-mids \rightarrow f(u,v), \frac{\partial^2 f(u,v)}{\partial u^2}, \frac{\partial^2 f(u,v)}{\partial v^2}, \frac{\partial^4 f(u,v)}{\partial u^2 \partial v^2}$$

$$v-mids \rightarrow f(u,v), \frac{\partial^2 f(u,v)}{\partial u^2}, \frac{\partial^2 f(u,v)}{\partial v^2}, \frac{\partial^4 f(u,v)}{\partial u^2 \partial v^2}$$

$$center \rightarrow f(u,v), \frac{\partial^2 f(u,v)}{\partial u^2}, \frac{\partial^2 f(u,v)}{\partial v^2}, \frac{\partial^4 f(u,v)}{\partial u^2 \partial v^2}$$

Therefore, given four corner points with those five values, we can create the *u*-midpoints, *v*-midpoints, and center point with the central differencing. From there, we can recurse into the four smaller squares formed by the new points to continue tessellation. Pseudo-code for doing this is shown in Listing 2. We start by explicitly computing this information for the four patch corner points, and then recurse. That's great, but there's still something missing. We've proved the recursive step, but we don't have a base case. When do we know we're done? How far do we recurse?

Recursion Heuristics

We already mentioned a very powerful tool for deciding when to stop recursing. The nonlinearity term from the central differences is an estimation of how curved the surface is at the spot we're tessellating. We can use the nonlinearity terms from the generation of the midpoints to determine, then, the curvature of the subpatch we're working with. Then,

FIGURE 3. *The nonlinearity vector lies roughly in a slice of the frustum.*

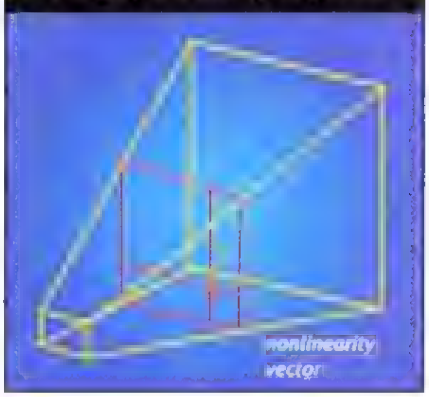


FIGURE 4. *The trigonometric derivation of the width of the frustum slice.*

given some threshold, if the magnitude of the nonlinearity terms for all the midpoints was less than the threshold, we'd just leave the subpatch as the two triangles formed by the corners and return. So if the threshold is very high, we end up with a very low-detail patch of just two triangles. If the threshold is too low, the patch is extremely detailed, which will look good but run slowly. Hence, picking a good threshold value is just a matter of testing, and would be a good candidate for an options screen for the game player.

However, just because a surface is very curvy doesn't mean we want it to continue tessellating. If the surface is 50 miles away from the camera and occupies a three-by-three block of pixels on the screen, we'd rather it be very low detail. However, if we zoomed in with a sniper rifle (think GOLDENEYE) to look at that terrain, it might be 50 miles away, but might now occupy a 200-by-200 block of pixels, in which case we'd like it to tessellate quite a bit. So, we need a way to add distance and field-of-view angle into our heuristic and we'd like it to be fast. Ideally, we'd like to know how big the nonlinearity vector is in pixels.

Here's a good, albeit rough, heuristic. Consider the camera frustum. The nonlinearity vector lies roughly in some slice of the frustum perpendicular to the screen (see Figure 3). Let's say we can figure out how wide, in pixels, that slice of frustum is. Then the ratio of the magnitude of the nonlinearity vector to the width of the frustum slice is the (very rough) size of the nonlinearity in screen space, expressed as a percentage of the screen width. For instance, if the patch was so curvy that the nonlinearity vector stretched all the way across that slice of frustum, the ratio would be 1. More likely, the nonlinearity vector will be something relatively small and the ratio will be something like 0.002, which is 0.2 percent of the screen, equal to about five pixels in a 640×480 screen display.

However, to do this, we still need to find the width of that frustum slice. So first we take the vector from the eye to a point near the nonlinearity vector, like one of the corner points. Call that vector *d*, for distance. Next we need the field-of-view angle, and then we're all set to do a little trigonometry. Referring to Figure 4, we know one angle of the triangle, and *d* is one of the legs, so the other leg is half the width of the frustum slice. So that leg's length is *d * tan(fov/2)*. Double that, and we have the width of the frustum slice. Then take the ratio of the nonlinearity vector's magnitude to the frustum slice width, and we have our heuristic.

Keep in mind that all of this is very rough. Since it's all about the visual quality, all of these heuristics are very open to tweaking. For instance, in my terrain demo, I decided that I cared more about detail of terrain close to the camera than far away, so I squared the distance attenuation part so it fell off faster and I could devote more tri-

LISTING 2. *This code takes four patch corners and a number of derivatives at those corners and finds the midpoints and center points.*

```
// Find the midpoints and center point given these four corners.  Any point has four
// values, u0v0 is the point (the zeroth derivatives w/ respect to u and v, u2v0
// is the second partial in u, u0v2 is the second partial in v, and u2v2 is the
// second partial in u of the second partial in v.
DifferencePatch(c0, c1, c2, c3)
{
  // Here are the points we're trying to find.
  points umid0, umid1, vmid0, vmid1, cen;

  // Find the u-midpoints' u0v0 and u2v0 through Central Differencing.
  umid0.u0v0 and umid0.u2v0 = CentralDifference(c0.u0v0, c0.u2v0, c1.u0v0, c1.u2v0);
  umid1.u0v0 and umid1.u2v0 = CentralDifference(c2.u0v0, c2.u2v0, c3.u0v0, c3.u2v0);

  // Find the v-midpoints' u0v0 and u0v2 through Central Differencing.
  vmid0.u0v0 and vmid0.u0v2 = CentralDifference(c0.u0v0, c0.u0v2, c1.u0v0, c1.u0v2);
  vmid1.u0v0 and vmid1.u0v2 = CentralDifference(c2.u0v0, c2.u0v2, c3.u0v0, c3.u0v2);

  // Find the u-midpoints' u0v2 and u2v2 through Central Differencing.
  umid0.u0v2 and umid0.u2v2 = CentralDifference(c0.u0v2, c0.u2v2, c1.u0v2, c1.u2v2);
  umid1.u0v2 and umid1.u2v2 = CentralDifference(c2.u0v2, c2.u2v2, c3.u0v2, c3.u2v2);

  // Find the v-midpoints' u2v0 and u2v2 through Central Differencing.
  vmid0.u2v0 and vmid0.u2v2 = CentralDifference(c0.u2v0, c0.u2v2, c1.u2v0, c1.u2v2);
  vmid1.u2v0 and vmid1.u2v2 = CentralDifference(c2.u2v0, c2.u2v2, c3.u2v0, c3.u2v2);

  // Now we just have to find the center point.  Find it from the midpoints.
  cen.u0v0 and cen.u2v0 = CentralDifference(vmid0.u0v0, vmid0.u2v0, vmid1.u0v0,
    vmid1.u2v0);
  cen.u0v2 and cen.u2v2 = CentralDifference(umid0.u0v2, umid0.u2v2, umid1.u0v2,
    umid1.u2v2);

  // We're done!
}
```

angles to the patches up front. In addition, while it's possible to sit down and hash out the ideal threshold value for different performance levels, you'll probably have more luck (and spend less time) just making the threshold modifiable at run time and tweaking it to suit your needs.

Filling in the Cracks

One of the dark secrets of this dynamic tessellation that we haven't mentioned is that it causes cracking in the patch. When two subpatches that share an edge are tessellated differently, there will be a hole between them, as shown in Figure 5. In practice, the cracking is very evident and distracting, so it can't just be passed off as an acceptable visual glitch.

At first glance, one might be tempted to go with a naive fix, like just filling the cracks with triangles. The problem is, if the higher-detail surface is, say, two levels of detail higher, the hole won't be triangular, but pentagonal. It turns out that it's easier and faster to fix the cracks in a better way than that.

Cracking is caused when subpatches that share an edge are of different levels of detail. Therefore, the fix for the problem is to make it so that shared edges are always at the same level of detail. In my terrain demo, I use a postpass over the vertices to accomplish this. For each patch, I check to make sure that the edges of its four subpatches are of the same level of detail. If one of them isn't at the same level of detail, I fix it by collapsing the middle vertex of the higher-detailed edge into the corner, as in Figure 6. Then I recurse on the four subpatches. While this does require another pass over the data, it works surprisingly well.

Now that we are tessellating patches dynamically with no cracking, we're getting closer. But we're not there yet. As far as we know, the patch is still a flat, unlit color. Next, we'd like to make it look a bit better.

Textures and Light Maps

One of the best ways to improve the visual quality of a patch is through texturing and lighting. The first thing we need are texture coordinates so we know how to place the texture onto the patch. The simplest approach, sufficient for many purposes, is just to use the patch (u,v) as the texture coordinates. We can get these easily from the central difference tessellation by averaging the (u,v) of the corners to get those of the midpoints and center point.

Given those, it's easy to drop a texture onto the patch. We just toss a terrain texture of choice at OpenGL, and then use the patch (u,v) at each point as texture coordinates. This maps the texture directly across the surface. But a texture with no lighting still doesn't look very good.

It might be tempting to use OpenGL's lighting as we did with the `UniformPatchTessellator`. However, there are a number of reasons not to do this. For one, finding the normal to the surface at each point requires a fair amount of work, including the two vector normalizations and cross product. Furthermore, the central differencing is already complicated enough without having to interpolate the normals. And there's one more reason not to use OpenGL lighting: OpenGL's per-vertex lighting makes the dynamic tessellation of our terrain very obvious. When the detail level of the terrain changes, the lighting shifts disturbingly, and when the polygon count in the terrain is low, the lighting looks stretched and linear from the Gouraud shading.

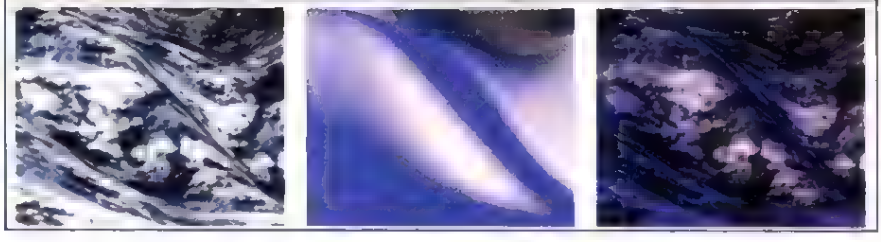
The solution is to use light maps. (For a more thorough discussion of light maps, see "Multitexturing in DirectX 6" by Jason Mitchell et al. [September 1998]). Light maps are handy not only because they not change with the tessellation, but they also actually give the illusion of more detail. Even if the surface is made of two triangles, the light map still depicts smooth, curved lighting playing across the curves that aren't really there.

In my terrain demo, I didn't particularly want to pre-calculate my light

FIGURE 5. *An edge shared by subpatches at different detail levels exhibits an open triangular hole. The z value of the extra point in the red patch doesn't line up with the blue patch's edge.*

FIGURE 6. *By collapsing the center vertex into the corner, the cracking is fixed.*

FIGURE 7. *Terrain Texture * Terrain Light Map = Lit Terrain.*



LISTING 3. *This code uses* `UniformPatchTessellator` *and OpenGL feedback mode to make a light map for the terrain.*

```
// We want a huge projection so that it won't clip anything, and we want
// it orthographic to save on transformation cost.
::glMatrixMode( GL_PROJECTION );
::glLoadIdentity();
::glOrtho( -10000, 10000, -10000, 10000, 1, 20000 );

::glMatrixMode( GL_MODELVIEW );
::glLoadIdentity();
::gluLookAt( 0,0,10000, 0,0,0, 1,0,0 );

::glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

// Position the light
::glEnable(GL_LIGHTING);
float position[] = {20.0f,0.0f,5.0f, 0.0f};
::glLightfv(GL_LIGHT0, GL_POSITION, position);

// Tessellate our patch
generatePoints( size, controls, basesU, basesV );

// Kick OpenGL into feedback mode and render the patch.
float* feedBuffer = new float[ size * size * 8 ];

::glFeedbackBuffer( size * size * 8, GL_3D_COLOR, feedBuffer );
::glRenderMode( GL_FEEDBACK );

::glColor3f( 1, 1, 1 );
::glBegin( GL_POINTS );
for ( int v = 0; v < size; v++ )
{
  for ( int u = 0; u < size; u++ )
  {
    ::glNormal3fv( norms + (u + (v*size))*3 );
    ::glVertex3fv( verts + (u + (v*size))*3 );
  }
}
::glEnd();

// Read the data out of the buffer
int count = ::glRenderMode( GL_RENDER );

int texPos = 0;
float* texData = new float[ 3 * size * size ];

for( int x = 0; x < count; x++ )
{
  if ( feedBuffer[ x ] == GL_POINT_TOKEN )
  {
    texData[ texPos + 0 ] = feedBuffer[ x + 4 ];
    texData[ texPos + 1 ] = feedBuffer[ x + 5 ];
    texData[ texPos + 2 ] = feedBuffer[ x + 6 ];

    texPos += 3;
    x += 7;
  }
  else
  {
    std::cout << "ERROR parsing feedback buffer array." << std::endl;
    delete[] texData;
    return 0;
  }
}
```

Continued on p. 47.

maps, and I didn't want to implement my own lighting model, either. Therefore, I found a handy solution that generates fairly good-looking results. OpenGL feedback mode.

The idea here is that at patch creation time, we want to sample the lighting at a number of points evenly spaced across the surface, and use their lit colors as texels in a light map. It turns out that even a fairly small light map will look just fine. Since we want the points evenly spaced, we'll use `niformPatchTessellator` again to generate an 8×8 grid of points the slow way, light them, and build a texture out of it. The code for this is shown in Listing 3. The process is surprisingly fast — a single patch's light map takes around 2 or 3 milliseconds to generate and upload to OpenGL.

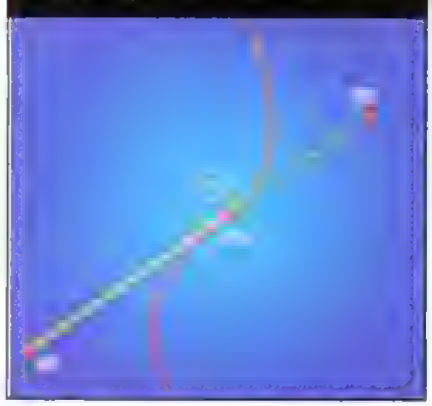
Figure 7 shows the end product. We have the textured pass, the lit pass, and when combined, the textured and lit patch. Quite an improvement, indeed. If I'd promised a terrain system, though, a single patch is hardly sufficient. We need to take this good-looking patch and somehow come up with a way to define an entire landscape.

Multiple Patches

Making terrain out of multiple patches doesn't sound like anything monumental. After all, we could just create a five-by-five grid of patches and draw them. Unfortunately, albeit not surprisingly, it's not that easy.

The first and most obvious problem is that if we just generate a 4×4 grid of random control points for each patch, the

FIGURE 8. *For the curves to connect smoothly, their endpoint tangents must both point in the same direction.*



LISTING 3. *(continued from page 46.)*

```
// Make it into a texture.
unsigned int texNum;
glGenTextures( 1, &texNum );
glBindTexture( GL_TEXTURE_2D, texNum );

// Set the tiling mode.  We don't want lightmaps to repeat, or we get odd borders.
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);

// Set the filtering.
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);

gluBuild2DMipmaps( GL_TEXTURE_2D, 3, size, size, GL_RGB, GL_FLOAT, texData );

delete[] texData;

return texNum;
```

patches will line up in x and y, but the z values of patches along shared edges will have nothing to do with each other. The terrain will have gaping holes where patches touch.

We could make sure that the patches share the same edge points so that there wouldn't be any surface breaks between them. However, that's still insufficient—the Bézier patch representation doesn't guarantee anything about continuity between patches. We have to manually make sure that we preserve a number of conditions or else the seams between patches will likely be sharply discontinuous, not at all like real terrain.

Continuity Conditions

We need to ensure that the tangent vectors of patches along shared edges are the same, so that the terrain will look smooth, even between patches. A property of Bézier patches is that the tangent vectors at the edge points are defined by the edge control points and the control points one level in. So the tangent vector in the u direction at the upper-right corner is the vector through the corner point from the point one to its left.

To preserve continuity between patches, we need to ensure that the for every control point along a shared edge, the tangent vectors are the same. The three points involved in those two tangents consist of the control point itself, plus the control points to either side of it. If we make sure that those three points lie on a line, we know that our terrain will be smooth.

Figure 8 shows an example of this concept. The tangent vectors of the joining curves pass through the two nearest control points of each curve. If we don't want a crease between the curve, both tangent vectors have to point in the same direction.

The terrain demo generates evenly spaced x and y values for a grid of control points. Then I generate z values using a fractal terrain algorithm. After that, I do a post-pass on the points to move some of them to make sure that this condition holds.

It's true that this continuity scheme is pretty limiting. For instance, if you have a patch surrounded by other patches, its entire border and all the points one in are completely defined by the other patches. That's every point in the patch! If we think of the terrain as a chessboard, where each square is a patch, then after we define all the black-square patches, the white-square patches are already completely defined. This does mean that it's harder to have very local control over the terrain, but it's still quite possible to make nice-looking terrain. There are different, more relaxed continuity conditions, but they're more complex than what I've discussed. A description of these conditions can be found in Faux and Pratt's book in the references at the end of this article.

With this implemented, we can generate an array of control points and make patches from it, and know that it won't have any sharp seams or visible edges. Hypothetically, this could be a complete terrain system: a large array of textured, lit patches. What we haven't yet discussed though, are the speed considerations. If we leave it like this, we'll be drawing every patch, even those offscreen, and while the tessellation and drawing are fast for a single patch, if we're drawing a couple hundred patches, the frame rate will be abysmal. We need some way of drawing only those patches that are visible.

Camera Frustum Culling

Last month, I mentioned that the convex hull of a Bézier patch's control points is a good bounding volume for the patch. Therefore, if we make a bounding box out of the control points by taking the minimum and maximum x, y, and z values, we can use the bounding box to cull the patch. We still need a way to tell whether the box is inside the camera frustum, though.

For my terrain demo, I use an object, `ClipVolume`, that takes information about the camera and builds six `Plane` objects out of it: the left, right, bottom, top, near, and far planes. Planes are capable of telling you whether a point is inside or outside of them. Therefore, if all the points of a bounding box are outside any one of the planes, the box is completely outside the frustum. So, at the beginning of the frame, I build a `ClipVolume` out of the camera by running through the patches and testing their bounding boxes against the `ClipVolume`. Those that pass are drawn.

It's simple, and definitely an improvement, but it's still not fast enough. The problem is that we have to touch every patch in the terrain each frame, so if we have an $N \times N$ grid of patches, the running time of our culling is $O(N^2)$. That's not ideal, because we don't want the number of patches to heavily affect the frame rate. We need a way to find the visible patches without testing every single patch's bounding box.

Quadtree Terrain Storage

As luck would have it, there are a number of ways to do this. I used a quadtree data structure for the terrain demo. A quadtree is a tree where each

node has four branches. In this case, the patches are the leaf nodes. Then, the nodes at the next level of the tree each contain a 2×2 group of patches. They're contained in 2×2 groups by the nodes at the next higher level, and so on. Eventually, we have our top node, which holds the whole tree.

Each node contains a bounding box, which is the box that contains all of its sub-nodes. This makes it easier for us to cull the terrain. We start at the top node, and if it's within the frustum, we recurse on each of its four branches. If any node is ever outside the frustum, we just return and don't bother considering any of its sub-nodes, as they're all outside the frustum, also. This way, we can reduce our running time for culling an $N \times N$ grid of patches to $O(\lg N)$, which is a pretty tremendous win over $O(N^2)$.

To build one of these structures, we start with the patches, the leaf nodes. They have their bounding boxes. Then, out of each 2×2 block of patches, we build a higher-level node. The node's bounding box is the box that contains each of the patches' bounding boxes; we just take the minimum and maximum x, y, and z from each of the four boxes to form its box. Once we have all of those second-level nodes, we use the same algorithm to generate third-level nodes from 2×2 blocks of second-level nodes. We continue until we only have one node, and that's our top node.

Now we've got a smooth, textured, lit terrain culling quickly against the camera and drawing correctly. We're almost done. There's one last glitch that we have to take care of.

Crack Fixing Revisited

We fixed the cracking that occurs within a patch, but unfortunately, cracking also occurs along the edge between two patches when one patch is tessellated to a different level from the other. Fortunately, inter-patch cracking is the same kind of phenomenon as the internal patch cracking, and we can use the same approach to fix it. When we've finished tessellating the patches we can see, we just have each patch check its right and bottom edges against the patches that share those edges.

There is a catch, though. We have to make sure that we fix the inter-patch cracking before we do the internal patch crack fixing. The reason for doing so is a little confusing at first. We know that when we do the inter-patch crack fixing, we can collapse edge vertices into other edge vertices. We also know that when we fix the internal patch cracking, we can collapse non-edge vertices into either other non-edge vertices or potential edge vertices. We know that internal crack fixing will never move an edge vertex, but it may collapse a vertex *into* an edge vertex. On the other hand, inter-patch crack fixing does move edge vertices.

What does this mean? Imagine an arbitrary vertex on the edge of a patch. We'll call it V. If we fix the internal cracks first, we might collapse a vertex into V. Now when we fix the inter patch cracks, we might collapse V into something else. When we do this, we leave the vertex that was collapsed into V hanging where V no longer is. This warps the patch and stretches a visible hole in the patch.

If we fix the inter-patch cracks first, we'll collapse V into a corner. Then, when we fix the internal cracks and we collapse that vertex into V, it will get sent to V's new, correct location.

Things Yet to Be Optimized

This explanation is certainly more difficult than the material presented in last month's article. Nonetheless, we're done. The terrain demo uses all of the things we've discussed, from central differencing, texturing, lightmapping, and crack fixing to quadtree storage and camera culling. What's more, it looks good and runs fast.

That's not to say that it's done. There's still plenty of work left to be done with the terrain system. For instance, the central differencing threshold can cause abrupt "popping" when the camera gets close enough to merit a higher level of detail. Perhaps a system where the change was gradual, using two thresholds, one where the detail began changing and one where it was done changing, could help. Linearly interpolating the size of the nonlinearity vector over that period would prevent abrupt changes in the terrain.

Another aspect to work on is code optimization. While the algorithms themselves are fairly fast, the code is written to be legible instead of speedy. Hand-tuning some of the functions, such as the `CentralPatchTessellator`'s tessellation function, could speed up the process.

Furthermore, the current system is a bit of a memory hog. It uses 2D arrays for all the terrain, even though much of the data is unused, depending on the level of the tessellation and how much of the terrain is visible at any given time. A better memory management system could speed the system up by increasing cache coherency, and allow larger terrain sets within reasonable memory limits.

Of course there are aesthetic improvements and expansions that are begging to be made – a physics system, some wildlife, or some ponds would add quite a bit. The code is available from my web site (see the reference at left), and I invite all those who are interested to download it and make any improvements they can think of.

Hopefully, between last month's article and this one, I've given a solid introduction to one of the Next Big Things in 3D game engines. Send questions, comments, or cool modifications to my code, to me via e-mail. ■

- Farin, Gerald. *Curves and Surfaces for CAGD, A Practical Guide.* New York: Academic Press, 1997.
- Faux, I.D. and M.J. Pratt. *Computational Geometry for Design and Manufacture.* Chichester, UK: Ellis Horwood, 1979.
- Garland, Michael and Paul Heckbert. "Surface Simplification Using Quadric Error Metrics." *Proceedings of SIGGRAPH* (1997): pp. 209-216.
- Mortenson, Michael E. *Geometric Modeling.* New York: Wiley Computer Publishing, 1997.
- Watt, Alan and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice.* New York: ACM Press, 1992.

- Thanks to Crack.com for releasing the full source, music, and texture library from their last project, GOLGOTHA.
- The terrain texture from Figure 7 was used from the GOLGOTHA textures.

The full source to the terrain demo is available from my web site at http://www.cs.dartmouth.edu/~bsharp/gdmag.

Miller Freeman Game Group Presents

THE GAME DEVELOPER SALARY SURVEY

Game DEVELOPER

Recruit, retain, and compare employees better with the first professional developer salary survey for North America developed with Market Perspectives, Inc. and sponsored in part by Virtual Search.

AREAS OF ANALYSIS: Base salary, bonus, and total compensation for over 10 job categories, including: Executive Management, Producers, Lead Programmers, Programmers, Programming Director, Game/Level Designers, 3D Artists, Executive Producers, Art Directors, Team Leaders/Managers, Animators, 2D Artists and Audio Engineers.

PRICE: Single User-$995* Multi-User (single company)-$4995*
Single Report (by job title)-$149*
AVAILABLE: 1998-Now 1999-May 1999
SHIPPING: Via Email-Microsoft Excel file, Password Protected

THE STATE OF GAME TECHNOLOGY

Produced by DFC Intelligence in conjunction with the Miller Freeman Game Group

DFC Intelligence

This critical look at the future game hardware, platforms, and 3D game technologies will help every vendor, publisher, developer and investor better understand the unique issues that are involved with developing today's flagship games and consumer hardware platforms. Includes critical analysis of and interviews with 35 top companies.

AREAS OF ANALYSIS: Present and Future Roadmap of Critical PC Game Hardware Segments, 3D Cards, Gaming Peripherals and API Technology Roadmap, Publishing and Development issues, In-depth Examination of Market Share among Critical Segments, Key Insights from Industry Leaders

PRICE: $2,000*
AVAILABLE: May 1999
SHIPPING: by USPS

THE U.S. MARKET FOR VIDEO GAMES AND INTERACTIVE ELECTRONIC ENTERTAINMENT

Produced by DFC Intelligence in conjunction with the Miller Freeman Game Group

DFC Intelligence

This semiannual guide to the interactive entertainment business is used by industry leaders as a business planning and reference tool for the executive level. This expansive report is a comprehensive method for keeping up with the explosive interactive entertainment market. 500+ pages, 100+ charts and diagrams.

AREAS OF ANALYSIS: State of the Industry (hardware and software), Industry Forecasts, Next Generation Console Systems, Video Game History, Video Game Systems, Industry Statistics, Development to Retail, The Consumer, Trends

PRICE: $2500*
AVAILABLE: March 1999
SHIPPING: USPS

PUBLISHER PROFILES REPORT

Assess the strategies and performance of the largest (by 1998 revenues) 40 game publishers (public and private) in North America, Europe, and Japan. This is a comprehensive look at their operations, financial, developer-relations and marketing strategies.

AREA OF ANALYSIS: Sales, Earnings, Income, Assets, Cash-Flow, Employees, Technical Strengths, Growth

PRICE: $995*
AVAILABLE: July 1999
SHIPPING: by USPS

*plus shipping, handling, and sales tax when applicable

ORDER FORM

FAX 415 905 2222 (credit card orders only) **OR MAIL** Miller Freeman Game Group Harrison Street, San Francisco CA 94107 USA **QUESTIONS?** e-mail us at gdcinfo@mfgame.com **for more information: http://www.mfgame.com/**

Name Company Title Signature
(required on all orders)

Street Address City State Province Zip / Postal Code Country

Phone ( ) Fax E mail

ITEM #	QUANTITY	TITLE	PRICE

SALES TAX NY 8.5% CA TN & TX 8.25% IL/KS NM NJ GA Canada 7%GST NB/NS/NF 8%HST

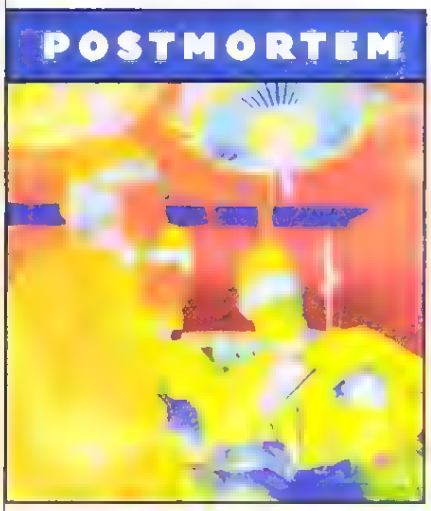
SUBTOTAL
SALES TAX
SHIPPING HANDLING
GRAND TOTAL

SHIPPING / HANDLING (EXCEPT SALARY SURVEY)
In U S $ 5.00
In Canada $ 8.50
Outside North America $ 20.00

FOR OVERNIGHT DELIVERY
Charge my Federal Express account #

FOR ORDERS IN U S DOLLARS FROM OUTSIDE NORTH AMERICA
Please prepay by credit card check drawn on U S bank bank transfer or request a pro forma invoice Orders less than US$250 payable only by credit card or bank transfer

METHOD OF PAYMENT
Payment enclosed (make checks payable to Miller Freeman Inc
Bank transfer (for orders from outside North America paid in US$)
Charge my Visa Mastercard American Express
Card # Exp date
Purchase order enclosed
Pro forma invoice (items will be shipped on receipt of payment



Postmortem: Looking Glass's THIEF: THE DARK PROJECT

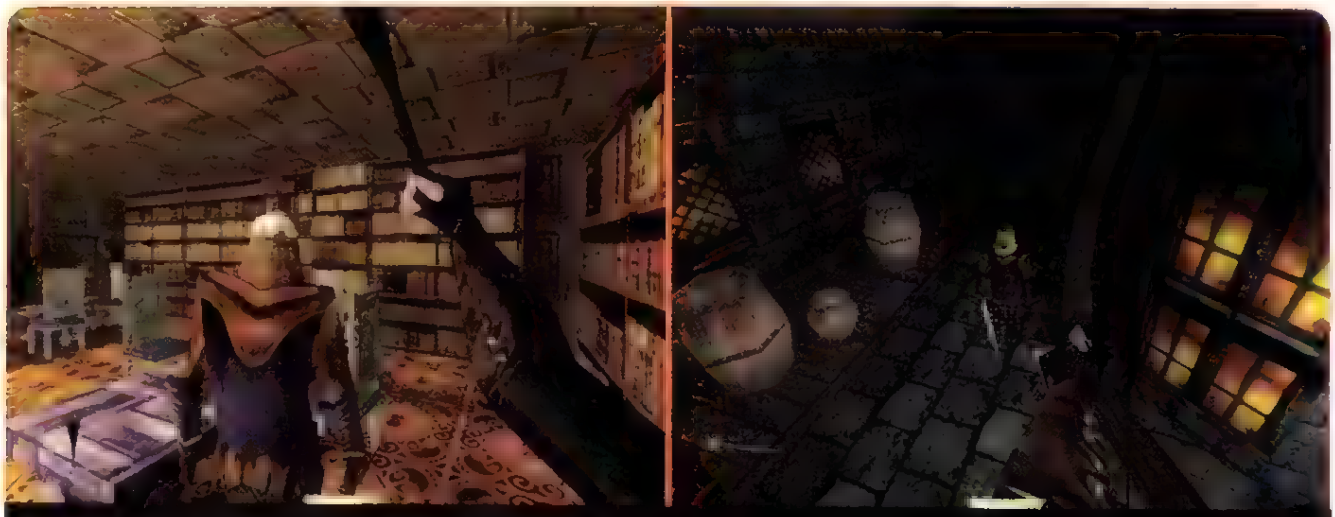
by Tom Leonard

THIEF: THE DARK PROJECT is one of those games that almost wasn't. During the long struggle to store shelves, the project faced the threat of cancellation twice. A fiscal crisis nearly closed the doors at Looking Glass. During one seven-month span, the producer, project director, lead programmer, lead artist, lead designer and the author of our renderer all left. Worse still, we felt a nagging fear that we might make a game that simply was not fun. But in the end, we shipped a relatively bug-free game that we had fun making, we were proud of, and that we hoped others would enjoy.

The Concept

The THIEF team wanted to create a first-person game that provided a totally different gaming experience, yet appealed to the existing first-person action market. THIEF was to present a lightly-scripted game world with levels of player interaction and improvisation exceeding our previous titles. The team hoped to entice the player into a deep engage-

Tom Leonard was the lead programmer for THIEF: THE DARK PROJECT, writing the AI and core architecture of the game. He lives in the Boston area. Tom has been at Looking Glass for three-and-a-half years, prior to which he spent seven years working on C++ development tools at Zortech and Symantec. He is currently working on next-generation technologies, and can be reached at tomlt@lglass.com.



Stealth is one of your best weapons in THIEF. The game's designers made sure that expert players would have to make effective use of silent weapons such as the blackjack and the bow and arrow.

ment with the world by creating intelligible ways for the world to be impacted by the player.

The central game mechanic of THIEF challenged the traditional form of the first-person 3D market. First-person shooters are fast-paced adrenaline rushes where the player possesses unusual speed and stamina, and an irresistible desire for conflict. The expert THIEF player moves slowly, avoids conflict, is penalized for killing people, and is entirely mortal. It is a game style that many observers were concerned might not appeal to players, and even those intimately involved with the game had doubts at times.

The project began in the spring of 1996 as "Dark Camelot," a sword-combat action game with role-playing and adventure elements, based on an inversion of the Arthurian legend. Although development ostensibly had been in progress on paper for a year, THIEF realistically began early in 1997 after the game was repositioned as an action/adventure game of thievery in a grim fantasy setting. Up to that point we had only a small portion of the art, design, and code that would ultimately make it into the shipping game. Full development began in May 1997 with a team comprised almost entirely of a different group of people from those who started the project. During the following year, the team created a tremendous amount of quality code, art, and design.

But by the beginning of summer in 1998, the game could not be called "fun," the team was exhausted, and the project was faced with an increasingly skeptical publisher. The Looking Glass game design philosophy includes a notion that immersive gameplay emerges from an object-rich world governed by high-quality, self-consistent simulation systems. Making a game at Looking Glass requires a lot of faith, as such systems take considerable time to develop, do not always arrive on time, and require substantial tuning once in place. For THIEF, these systems didn't gel until mid-summer, fifteen months after the project began full development, and only three months before we were scheduled to ship.

When the game finally did come together, we began to sense that not only did the game not stink, it might actually be fun. The release of successful stealth-oriented titles (such as METAL GEAR SOLID and COMMANDOS) and more content-rich first-person shooters (like HALF-LIFE) eased the team's concerns about the market's willingness to accept experimental game styles. A new energy revitalized the team. Long hours driven by passion and measured confidence marked the closing months of the project. In the final weeks of the project the Eidos test and production staff joined us at the Looking Glass offices for the final push. The gold master was burned in the beginning of November, just in time for Christmas.

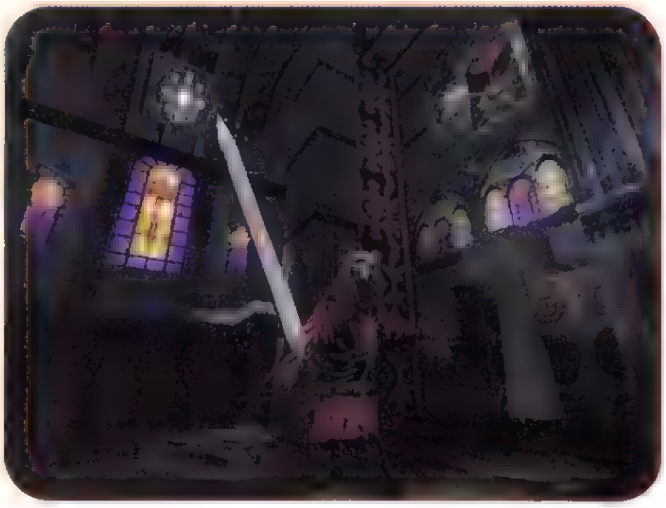
In many ways, THIEF was a typical project. It provided the joys of working on a large-scale game: challenging problems, a talented group of people, room for creative expression, and the occasional hilarious bug. It also had some of the usual problems: task underestimation, bouts of low morale, a stream of demos from hell, an unrealistic schedule derived from desire rather than reality, poor documentation, and an insufficient up-front specification.

However, THIEF also differed from a number of projects in that it took risks on numerous fronts, risks that our team underappreciated. We wanted to push the envelope in almost every element of the code and design. The experimental nature of the game design, and the time it took us to fully understand the core nature of that design, placed special demands on the development process. The team was larger than any Looking Glass team up until then, and at

THIEF: THE DARK PROJECT

Looking Glass Studios Inc.
Cambridge, Mass.
(617) 441-6333

Release date: December 1998
Intended platform: Windows 95/98
Project budget: Aproximately $3 million
Project length: 2.5 years
Team size: 19 full-time developers. Some contractors.
Critical development software: Microsoft Visual C++ 5.0, Watcom C++ 10.6, Opus Make, PowerAnimator, 3D Studio Max, Adobe Photoshop, AntimatorPro, Debabilizer, After Effects, and Adaptive Optics motion-capture processing.



more than 50 people worked in one way or another on THIEF some as part of the "Camelot" project, others as part of the Looking Glass audio-visual and technology support staff, some as helpful hands from other Looking Glass projects. The core team consisted of a number of veterans of previous Looking Glass titles (UNDERWORLD I and II, SYSTEM SHOCK, FLIGHT UNLIMITED, TERRA NOVA, BRITISH OPEN CHAMPIONSHIP GOLF, and the unpublished STAR TREK: VOYAGER) as well as some new industry arrivals. The project had a number of very talented people and strong wills. Although it took some time for the team to unite as a tight-knit creative force, the final six months were incredibly productive, spirited, and punishingly fun.

times there seemed to be too many cooks in the kitchen. Reaching a point where everyone shared the same vision took longer than expected. A philosophy of creating good, reusable game engine components created unusual challenges that didn't always fit well with schedule and demo pressures. The many risks could have overwhelmed the project, if not for the dedication creativity, and sacrifices of the team.

Throughout the life of the project,

What Went Right

1. DESIGNING DATA-DRIVEN TOOLS. Our experience on previous titles taught us that one of the impediments to timely game development is the mutual dependence of artists, designers, and programmers at every development stage. One of the development goals for the Dark Engine, on which THIEF is built, was to create a set of tools that enabled programmers, artists, and designers to work more effectively and independently. The focus of this effort was to make the game highly data-driven and give non-programmers a high degree of control over the integration of their work. Media and game systems were to be easily and intuitively plugged in and edited by the team members responsible for their creation, without requiring the direct involvement of programmers.

The Dark Object System stood at the heart of our strategy. Primarily designed by programmer Marc "Mahk" LeBlanc, the Object System was a general database for managing the individual objects in a simulation. It provided a generic notion of properties that an

object might possess and relations that might exist between two objects. It allowed game-specific systems to create new kinds of properties and relations easily, and provided automatic support for saving, loading, versioning, and editing properties and relations. It also supported a game-specific hierarchy of object types which could be loaded, saved, and edited by designers. Programmers specified the available properties and relations and the interface used for editing, using a set of straightforward classes and structures (using GUI Tools), the designers specified the hierarchy and composition of game objects, independent of the programming staff. In fact, there was no code-based game object hierarchy of any kind.

Although the implementation of the system was much more work than we expected, and management of the object hierarchy placed significant demands on lead designer Jim Stolmach, it turned out to be one of the best things in the project. Once the set of available properties and relations exposed by programmers was mature, the Object System allowed the designers to specify most of the behaviors of the game without scripting or programmer intervention. Additionally, the relative ease with which variables could be made available to designers in order to tweak the game encouraged programmers to empower the designers thoroughly.

The second major component of our strategy was our resource management

Hand-to-hand combat is sometimes necessary.

and flying out of the room
LocoMotion
studios
the fine art of motion capture
108 South Valley View • Wimberley,Texas • 78676 • 888•755•7977

system. The resource management system gave the game high-level management control of source data, such as texture maps, models, and digital sounds. It helped manage the game's use of system memory, and provided the data flow functions necessary for configuration management.

Looking Glass's previous resource management system provided similar functionality, but identified resources by an integer ID and required a special resource compilation step. This technique often required recompilation of the game executable in order to integrate new art, and required that the team exit the game when resources were published to the network. The new system referred to a resource by its file name without its extension, used a file system directory structure for namespace management, didn't leave files open while working, and required no extra compilation step. Developers simply dropped art into their local data tree and started using it. To expose art to the rest of the team, lead artist Mark Lizotte just copied art into the shared project directories. Compound resources were treated as extensions to the file system and were built using the standard .ZIP format. This allowed us to use off-the-shelf tools for constructing, compressing, and viewing resource files. The system facilitated content development by allowing programmers, artists and designers to add new data to an existing game quickly.

The data-driven approach worked so well that through much of our development, THIEF and SYSTEM SHOCK 2 (two very different games) used the same executable and simply chose a different object hierarchy and data set at run time.

2. SOUND AS A GAME DESIGN FOCUS. Sound plays a more central role in THIEF than in any other game I can name. Project director Greg LoPiccolo had a vision of THIEF that included a rich aural environment where sound both enriched the environment and was an integral part of gameplay. The team believed in and achieved this vision, and special credit goes to audio designer Eric Brosius.

As an element of the design, sound played two roles in THIEF. First, it was the primary medium through which the AIs communicated both their location and their internal state to the player. In THIEF we tried to design AIs with a broader range of awareness than the typical two states that AIs exhibit, "oblivious" and "omniscient." Such a range of internal states would be meaningless if the player could not perceive it, so we used a broad array of speech broadcast by the AIs to clue in the player. While very successful for humanoid AIs, we feel the more limited expressibility of non-human creatures is the heart of why many customers didn't like our "monster levels."

Second, the design used sounds generated by objects in the game, especially the player, to inform AIs about their surroundings. In THIEF, the AIs rarely "cheat" when it comes to knowledge of their environment. Considerable work went into constructing sensory components sufficient to permit the AIs to make decisions purely based on the world as they perceive it. This allowed us to use player sounds as an integral part of gameplay, both as a way that players can reveal themselves inadvertently to the AIs and as a tool for players to distract or divert an AI. Moreover, AIs communicated with each other almost exclusively through sound. AI speech and sounds in the world, such as the sound of swords clashing, were assigned semantic values. In a confrontation, the player could expect nearby AIs to

Concept sketch of Hammer.

Concept sketch of a burrick.

Take five years of *Game Developer* for a spin.

Watch your projects accelerate.

Hard-to-find technical information is all yours —all in one place

You've depended on *Game Developer* to help you make better games, but do you have ALL the gems the magazine has provided over the years? Many back issues are out of print. But now the entire history of *Game Developer* from 1994-April 1999 is available on one great resource. On the *Game Developer* CD-ROM, you'll find quality, cut-to-the-chase information for professional game developers that you just can't get anywhere else. Imagine the different approaches you'll take after reading all 18 Postmortem columns that dissect the experiences of developers working on published games. Take advantage of years of shared wisdom from experts like Chris Hecker on programming, Paul Steed and Josh White on art, and Mark Miller on audio.

- **Solutions** right on your desktop with the full text of feature articles, columns, and product reviews from the entire *Game Developer* archive (40 issues)
- **Visual clarity** — articles, artwork, and diagrams appear exactly as they were printed in the magazine
- **Easy to use** across all platforms — thanks to the Adobe Acrobat format
- **Source code** from each issue accompanies the articles.

Order this unique CD-ROM today!

Quantities are limited. Seriously.

Only $99 plus $5 shipping and handling in the U.S. Add sales tax where applicable.

Here's how to order:

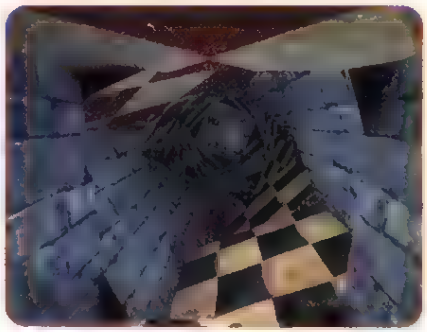
CALL 1-800-444-4881

FAX 785-841-1631 785-841-2624

ONLINE www.gdmag.com/cdrom

EMAIL orders@mfi.com MAIL TO: *Game Developer* CD-ROM, 1601 West 23rd Street, Suite 200, Lawrence, KS 66046-2703 USA

Game DEVELOPER



become alarmed by the sound of combat or cries for help, and was thus encouraged to ambush opponents as quietly as possible.

In order for sound to work in the game as designed, we needed to implement a sound system significantly more sophisticated than many other games. When constructing a THIEF mission, designers built a secondary "room database" that reflected the connectivity of spaces at a higher level than raw geometry. Although this was also used for script triggers and AI optimizations, the primary role of the room database was to provide a representation of the world simple enough to allow realistic real-time propagation of sounds through the spaces. Without this, it is unlikely the sound design could have succeeded, as it allowed the player and the AIs to perceive sounds more as they are in real life and better grasp the location of their opponents in the mission spaces.

3. FOCUS, FOCUS, FOCUS. Early on, the THIEF plan was chock full of features and metagame elements: lots of player tools and a modal inventory user interface to manage them; multiplayer cooperative, death-match and "theft-match" modes; a form of player extra-sensory perception; player capacity to combine world objects to create new tools; and branching mission structures. These and other "cool ideas" were correctly discarded.

Instead, we focused in on creating a single-player, linear, mission-based game centered exclusively around stealth, with a player toolset that fit within the constraints of an extension of the QUAKE user interface. The notion came into full force with two decisions we made about seven months before we shipped. First, the project was renamed THIEF from the working title "The Dark Project," a seemingly minor decision that in truth gave the team a concrete ideological focus. Second, we decided preemptively to drop multiplayer support, not simply due to schedule concerns, but also to allow us as much time as possible to hone the single-player experience. In the end some missions didn't achieve the stealth focus we wanted, particularly those originally designed for "Dark Camelot," but the overall agenda was the right one.

4. OBJECTIVES AND DIFFICULTY. One of the THIEF team's favorite games during development was GOLDENEYE on the N64. We were particularly struck by the manner in which levels of difficulty were handled. Each level of difficulty had a different overlapping set of objectives for success, and missions were subtly changed at each level in terms of object placement and density. Relatively late in the development of THIEF, we decided such a system would work well in our game. Extending the concept, we added a notion that as difficulty increased, the level of toleration of murder of human beings decreased. We also allowed players to change their difficulty level at the beginning of each mission. The system was a success in two ways. First, it made clear to the player exactly what "difficulty" meant. Second, it allowed the designers to create a very different experience at each level of difficulty, without changing the overall geometry and structure of a mission. This gave the game a high degree of replayability at a minimum development cost.

5. MULTIPLE NARROW-PURPOSE SCRIPTING SOLUTIONS. Although the Object System provided a lot of flexibility, we also needed a scripting language to fully specify object behaviors. Rather than create a single all-encompassing scripting system, we chose to develop several more focused tools for scripting. This tiered scripting solution worked well.

In creating our core "high-end" object scripting technology, we wanted to allow designers with moderate programming skill to create complex object behaviors easily. Scripts were event-driven objects attached at run time to game objects, and contained data, methods, and message handlers. The game provided a selection of services to allow the script to query the world state and the game object state, and also to perform complex tasks. Our goal was to create a scripting language that offered source-level debugging, was fast, and was dynamic. The solution was essentially C++ in DLLs, compiled by the C++ compiler, using a combination of classes and preprocessor macros to ease interface publishing, handle dynamic linking, and provide designers a clear programming model. Though used by both programming-savvy designers and programmers, the fact that it was a real programming language prevented widespread use by all of the designers.

Most designers were interested in customizing AI behaviors. For the AI we created a simpler scripting system, "Pseudo-scripts," that were implemented as properties within the Object System. Pseudo-scripts took the burden of coding scripts off of the designers. The AI provided a stock set of triggers, such as "I see the player near an object" or "I see a dead body"; the designer provided the consequence of the trigger. Each Pseudo-script was edited in a dialog box presenting parameters to tweak the "if" clause of the trigger, and space for a list of simple, unconditional actions to perform when the trigger fired. In this way, the custom behavioral possibilities of the AI at any moment were described by the aggregate of Pseudo-scripts that were attached to that AI. This approach had three benefits. First, it was simple enough so that designers with no pro-

gramming experience were comfortable using it. Second, it narrowed the range of triggers a designer could use to a good pre-selected set, rather than giving them an open-ended system that might not have worked as well. Finally, when and how to evaluate AI triggers, a potential run-time expense if not carefully constructed, could be custom built by a programmer.

The final scripting system built into THIEF was the Tagged Schema system. When the game required motions and sounds, it requested them as concepts modified by optional qualifiers, rather than directly. For example, an AI who had just heard the player would request the concept "moderate alert" qualified with an optional tag like "+sense:sound." A potential set of resources was then chosen using a pattern matcher; in this example, it would choose all samples in that AI's voice expressing a generic "something's not right," all samples expressing "I heard something fishy," but no samples expressing "I saw something fishy." From this set, the specific resource was then chosen using a weighted random selection. The tables used were specified by the designers using a simple language. Specifying motion and sound selection this way, designers created an interesting variety of randomized environments and behaviors without changing the code of the game.

What Went Wrong

1. TROUBLE WITH THE AI. If one thing could be called out as the reason THIEF's gameplay didn't come together until late in the process, it would be the AI. The AI as a foil to the player is the central element of THIEF, and the AI we wanted wasn't ready until late in the spring of 1998. As lead programmer and author of the final AI, I take full responsibility for that.

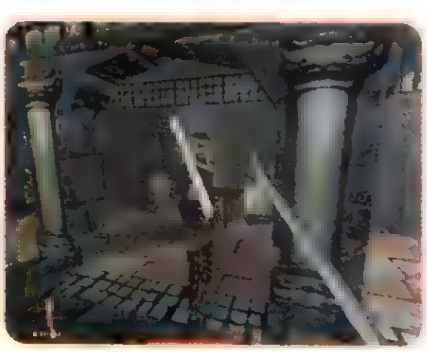
The original AI for THIEF was designed by another programmer before the requirements of the revised stealth design were fully specified. Six months after it was begun, the project director and overseer of the system left the team, and the most of the programming staff was temporarily reassigned to help ship another game that was in trouble. During the following months development on that AI continued without any oversight and without a firm game design. Soon after, the programmer working on the AI also left. While the core pathfinding data structures and algorithms were basically sound, the code that generated the pathfinding database was extremely buggy. The design of the AI decision process was geared towards an action fighting game requiring little designer customization, rather than a stealth game that needed much more customization. Even worse, the high-level decision process in the AI had drifted away from a rigorous design and the code was extremely brittle. The whole situation was a disaster.

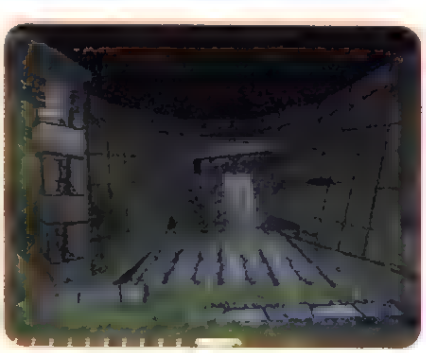
These might not have been serious issues, except for one key mistake: I didn't realize the depth of the problem quickly enough, and despite concerns expressed by programmer/designer Doug Church, I didn't act fast enough. I think highly of the programmer involved with the initial AI and wanted to avoid the natural but often misguided programmer reaction within myself that I should just rewrite it my way. So, I took the position that while buggy, the system as a whole was probably sound. Several months and many sleepless nights later, I concluded that I had been sorely mistaken.

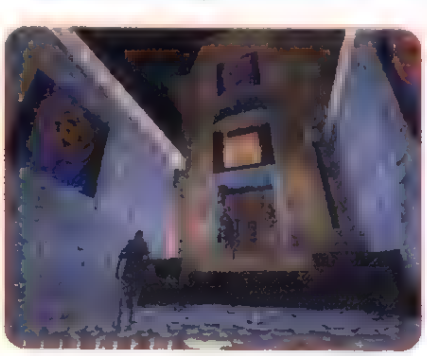
By November 1997, I had the basics of a new design and began working on it. But as I work had to stop in order to pull together an emergency proof-of-concept demo by the end of December to quell outside concerns that the team lacked a sound vision of the game. This turned into a mid-January demo, followed by an early February publisher demo, followed by a late February make-or-break demo. During this time the only option was to hack features as best we could into the existing AI. While better than losing our funding, constructing these demos was not good for the project.

In the end, work on the new AI didn't begin until mid-March. Despite the fact that our scheduled ship date was just six months away, we threw away four-fifths of our existing AI code and started over. After a hair-raising twelve-week stretch of grueling hours, the AI was ready for real testing. Had I committed to a rewrite two months earlier the previous autumn, I believe the AI would have been ready for real use three to five months sooner.

2. AN UNCERTAIN RENDERER. The project was started because of the renderer, rather than the reverse. The basic core of the renderer for THIEF was written in the fall of 1995 as an after hours experiment by programmer Sean Barrett. During the following year, the renderer and geometry-editing tools were fleshed out, and with "Dark Camelot" supposed to ship some time in 1997, it looked like we would have a pretty attractive game. Then, at the end of 1996, Sean decided to leave Looking Glass. Although he periodically contracted with us to add features and we were able to add hardware support and other minor additions, the renderer never received the attention it needed to reach the state-of-the-art in 1998. The possibility that we might not have a point programmer for the renderer weighed heavily on the team. Fortunately, Sean remained available on a contract basis, and other members of the team developed sufficient







One of the featureed weapons is the fire arrow.

knowledge of the renderer so that we shipped successfully. In the end, we shipped a renderer appropriate for our gameplay, but not as attractive as other high-profile first-person titles.

This may prompt the question of why we didn't simply license a renderer. When the project started a few months into 1996, the avalanche of QUAKE licenses hadn't really begun and UNREAL was still two years away. By the time licensing was a viable choice, the game and the renderer were too tightly integrated for us to consider changing.

3. LOSS OF KEY PERSONNEL AMID CORPORATE ANGST BEYOND OUR CONTROL. Midway through 1997, THIEF was just starting to gather momentum. We were fully staffed and the stealth design was really starting to get fleshed out. Unfortunately, Looking Glass's financial situation was bleak. Few emotions can compare to the stress of heading to work not knowing who might be laid off, including yourself, or whether the doors would be locked when you got there. The company shed half of its staff in a span of six months, and while the active teams tried to stay focused, it was hard when one day the plants were gone, another day the coffee machine then the water cooler.

Some of the THIEF team couldn't continue under these conditions. We lost two programmers, including the former lead programmer, and a designer. When we were forced to close our Austin office, we lost our producer, Warren Spector, as well as some programmers who made valuable technology contributions to our engine. All of these individuals are now on Ion Storm's DEUS EX team. Although it took some months to fully restore the spirit of the rest of the team, we held together and the company eventually rebounded. Perhaps it bestowed a stoicism that comes from knowing that however bad things might seem, you've already seen worse.

4. UNDERVALUED EDITOR. One of the boils never lanced on the project was our editor, Dromed. Although it was sufficiently powerful and provided the essential functionality we needed to ship the game, Dromed was a poorly documented and sometimes disagreeable editor. Dromed was first developed as a demonstration editor when the target platform of the game was DOS. As a demo, it never received the kind of formal specifications and designs one would expect for the central experience of the design team. As a DOS application, it lacked the consistent and relatively easy-to-use user-interface tools of Windows. An early mistake was our failure to step back and formally evaluate the editor, and then rebuild it based on our experience constructing the demo editor. We also should have designed a proper editor framework, and hired a dedicated Windows user-interface programmer to support it through development. In retrospect, the time lost cleaning up the editor probably would have been saved on the back end of the project.

5. INADEQUATE PLANNING. Although it is a cliché in the software industry to say our scheduling and budget planning were woefully inadequate, the THIEF project suffered greatly from this malady. There were several elements to our deficient planning.

During "Dark Camelot," and continuing through the first half of THIEF, we staffed the team before the design and technology was sufficiently mature. In THIEF, this led us to rush towards finishing the design, when we didn't necessarily *understand* the design and technology. With insufficient specifications of both the code systems and mission designs, we ended up doing lots of content that was essentially wrong for the game we were making. Code was written and spaces were built that weren't well directed towards the goals of the project.

To make matters worse, we failed to reassess core scheduling assumptions carefully once the schedule began to slip. Captives of a series of unrealistic schedules, we didn't leave enough time for the sort of experimentation, dialogue, and prototyping a project like THIEF needs. Late in the winter of 1998, many of our scheduling mistakes had been corrected. Still, during the remainder of the project, the legacy of our earlier missteps required cutting missions that relied on technology we didn't have, and reworking missions not focused on the core gameplay.

Stepping Back from the Project

THIEF was constructed as a set of appropriately abstract reusable game components designed for creating object-rich, data driven games. Although increasing the cost of development, this approach allowed Looking Glass to leverage various technologies across disparate types of games, from the first-person action game SYSTEM SHOCK 2 to our combat flight-simulator FLIGHT COMBAT. In our next generation technology, some of the systems, such as the AI and the Object System, will merely be revised, not rewritten. We intend to continue with this development philosophy in our future games.

The next time around, our approach to constructing the engine will differ. The engine will be scheduled, staffed and budgeted as a project in its own right. The editor will be treated as more of a first-class citizen than was the case in THIEF. Finally, a content development team will not be geared up until the technology is sufficiently mature to allow for an informed game design process.

Oh, and we'll get our schedules right — really. ■

WIN BUSINESS AT EVERY LEVEL!

ECTS gives you

- Big names from across the industry
- Publishers, developers, retailers, buyers, manufacturers, distributors, investors and media from all over Europe and the world
- Exclusive previews of all the very latest products and technologies
- Hundreds of product launches

The state of the art business forum for the entire leisure software industry.

Explore ECTS – and win!

ECTS

Europe's premier interactive entertainment expo

5 - 7 September 1999, London Olympia

Sponsored by

CTW

Miller Freeman

Register online NOW!

www.ects.com

or call the hotline for more information on +44 (0)1203 426 462

ON THE FRONT LINE OF GAME INNOVATION

Game DEVELOPER

GAME DESIGNERS

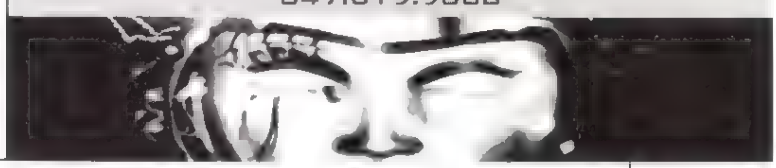
ANIMATORS

GRAPHIC DESIGNERS

MULTIMEDIA
PRODUCERS

SOFTWARE
DEVELOPERS

Reach these prime prospects with the Game Developer mailing list. Please contact Kristen Kohnke at Rubin Response Management Services, Inc. for complete details -
847.619.9800



Rubin Response Management Services, Inc.
Technology List Division • (847)619-9800 • Fax (847)619-0149
E mail kristenk@rubinresponse.com

ROGER WILCO™!

Resounding Technology offers fully customizable solutions for integrating real-time multi-point voice into your on-line games.

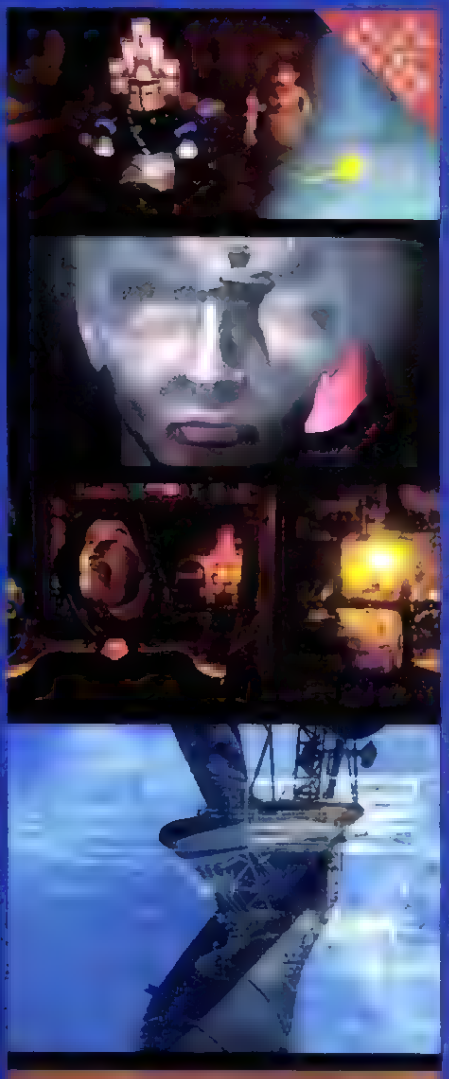
CUSTOM MULTIPLAYER VOICE COMMUNICATION SOLUTIONS

ROGER WILCO™ – AN INTERNET VOICE RADIO FOR MULTIPLAYER ONLINE GAMES

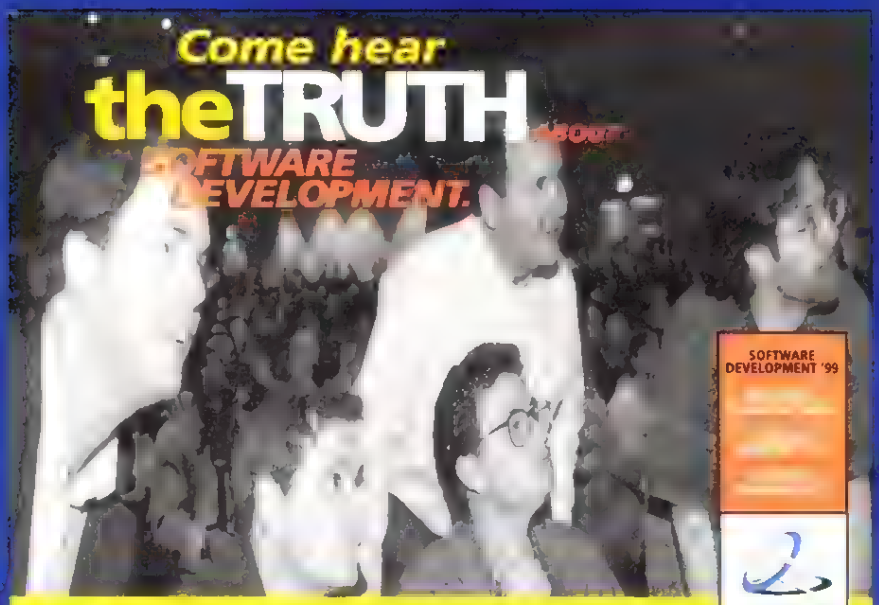
- No Servers
- Up to 128 People Per Channel
- Optimized for 28.8 Modems

Roger Wilco™
Resounding Technology
The Malden Campus
14 Dartmouth Street, Suite D
Malden, MA 02148-5104

www.rogerwilco.com
(781) 388-2670



Come hear theTRUTH about SOFTWARE DEVELOPMENT.



SOFTWARE DEVELOPMENT '99

SD 99

GET THE BIG PICTURE

THEN CHALLENGE THE INDUSTRY EXPERTS WHO CREATED IT.

The thing that makes Software Development 99 so exciting is the very same thing that makes it different from any other conference.

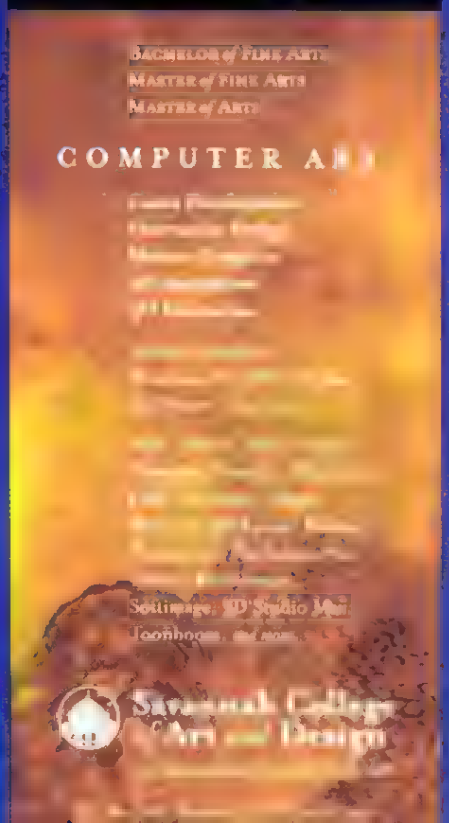
SD 99 is the only major forum for development professionals that is truly independent. Instead of a single point of view, you hear industry leaders representing a variety of technologies and a spectrum of competing visions. Then you evaluate and extract the truth for yourself.

SD 99 isn't a marketing event. It's about keeping developers ahead of the curve. We've rounded and more effective. Which is why, unlike any other, this conference has flourished for over a decade.

This year, choose from over 140 classes and tutorials on Java, Web, Internet development, Objects and Components, Modeling Techniques, Usability Issues, Management Practices and more. And test drive the latest tools from over 200 vendors.

So join thousands of developers and their managers at the most critical event in the industry – the one that focuses on developing the developer.

call for registration information (800) 441-8826 email us at sd99@mfi.com or visit www.sdexpo.com



ADVERTISER INDEX

NAME	PAGE	NAME	PAGE
ATI Technologies	7	Okino Computer Graphics	28
Conitec Datensysteme GmbH	62	Rad Game Tools, Inc.	C4
Duck Corporation	8	Resounding Technology	61
Immersion Corporation	19	Savannah College of Art and Design	61
Intel	3	SGI	11
Locomotion Studios	52,53	SGI	13
Metrowerks, Inc.	25	Stormfront Studios	28
Motion Analysis	C3	Trymedia	27
Multigen	15	Yamaha	4
Numerical Design Ltd.	C2		

Create cool 3-D Applications!
3D GameStudio
Authoring System for 3-D Realtime Applications
No Royalties!!
◆ 3-D applications without programming skills!
◆ Contains World Editor and the ACKNEX 3-D engine
◆ Free 3-D template game with 150+ textures included
◆ 3-D polygonal landscapes with slopes, bridges, buildings
◆ Player can walk, drive, fly, climb, swim, dive...
◆ User-defined panels, overlays, cockpits, and scrolling text
◆ Create your own objects, actors, walls, weapons, menus
◆ Resolution 320x400 (lite) up to 800x600 (professional)
◆ 8-channel stereo sound & midi; FLIC player (professional)
◆ Imports PCX, LBM, MDL, WAV, MID and IBK files
◆ Supports both DOS and Windows 95/98 with DirectX 5
◆ 200+ pages English manual with game tutorial
3D GameStudio Lite ... $99
3D GameStudio Commercial ... $199
(+SVGA+Multiplayer Modem/Network+2-Player Split-Screen mode)
3D GameStudio Professional ... $1150
(+Polygonal Actors+2-Player Modem/Network+CD-Audio+FLIC)
Infos, demos, user forum → http://www.conitec.com
CONITEC
1951 4th Ave, Ste 301 ● San Diego CA 92101
Tel (619) 702-4420 ● Fax 702-4419 ● www.conitec.com

Continued from page 64.

here on planet Earth are also responsible for the other 80 percent of a game's audio, which includes the dialogue, sound effects, interface feedback, and so on. The longer we promote ourselves as musicians, the harder it's going to be for us to work our way into the upper ranks of the team, which is where we must be if we are going to push audio design through to the next level. We need to be technologists, innovators, and sonic architects, not bong-smoking, noodle-headed, electronic music geeks.

THE HOLLYWOOD BIG WIG. There is also a large number of linear media post-production sound designers who work amongst us. While many of these professionals are extremely talented when it comes to complementing a fixed asset, like a film, with sound effects, dialogue and music, they have limited expertise in creating an interactive soundscape. Programmers spend a great deal of time thinking universally about all aspects of the world they will be creating, and we also need to think well beyond the sonic content we create.

There was at least one GDC session per day that was sprinkled with a heaping dash of crow about how the "sound design didn't turn out like I wanted it to." The audio professionals always point their fingers at the programmers and producers for this shortcoming, which seems like a pathetic cop-out to me. When was the last time you provided a code sample with your spec? Every piece of media on your list should have a complete description of how it interacts with all of its dependencies: user input, artificial intelligence, and so on. Leave no stone unturned, do your damn homework, take control of your world because if you don't, no one will. Waxing ecstatic about design concepts without being able to provide the developer with concrete examples of how it can be implemented is negligent and can be offensive to someone who has spent the last year thinking about everything *but* the audio. Educate yourself on basic programming concepts, understand the relationship between the sound in the game and everything it touches, and be a diplomat when it comes to working with the rest of the team, because they are holding your design in their hands.

I believe that we are only as influential as we want to be, because audio programming can be the most emotional and subliminal (read: core) information a videogame can deliver. However, if we continue to play the role of the "audio guy," we will constantly be ruled by program managers, producers, and developers, who all consider themselves to be critical to the product's success. If you start showing up to the table with the right cards in your hand, I can guarantee that your budget will grow, you'll break more bread, and the end product will bear a much greater resemblance to your original version. ■

Dear Game Developer,

If you're technically astute and have a way with words, Game Developer magazine needs you. We're looking for feature articles on graphics programming, testing and QA, producer/management issues, audio, art and animation, internet game development, new technologies, and game design. We're also looking for people to review game development tools.

Please send your abstract or outline to:
adunne@compuserve.com

Writer's guidelines can be downloaded from our web site at:
http://www.gdmag.com/writguid.htm

Thanks,

Editor in Chief

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

by Matt Johnston

Don't Call Me "Audio Guy"

While attending the recent Game Developers' Conference in San Jose, I left each audio session feeling like the little teapot in that nursery rhyme, except that I was not short and stout, I was hot, steaming and pissed off. At one of the audio sessions, a well-respected audio designer began his presentation in a room full of budding young producers with the classic, "I'm an audio guy and get no respect in this industry" inferiority complex. I've caught myself falling into this same trap in meetings and such, but propagating this impression throughout the gaming community is a big mistake. We need to be proactive contributors to our community. We need to assert confidence and power if we are going to be respected as equals.

There are three major stereotypes that are keeping us down, yet we regularly embody them in public:

THE "AUDIO GUY." I am sick of people referring to themselves as the "audio guy." When I think of an "audio guy," I think of someone who crawled off of the Van Halen *1984* tour. After destroying the remainder of his hearing doing live sound to support his cocaine habit, he landed a "day job" in the videogame industry. As you laugh, keep in mind that these people do exist in our industry and they are all around you. (Oh, and by the way, we're not all guys. While the traditional linear media post-production scene has been a pickle convention for some time now, the videogame industry contains some of the most talented female audio professionals in the world.)

Computer gaming since 1983 and a charter member of the Ted Nugent Bowhunting Association, Matthew Lee Johnston is now a Senior Audio Designer at Microsoft. He can be reached at mattj@microsoft.com

At a GDC audio session, I asked a question that made some people uncomfortable. I wanted to know why all current audio tools are designed with control surface metaphors that were innovated in or before the late 1950s (such as volume sliders, piano rolls, knobs, and *VU* meters). An "audio guy" actually had the audacity to tell me after the session that "you young guys just need to sit back and wait your turn." I think that his main fear was that he will someday have to become part programmer and part audio guy. The truth is, *he better*, because the videogame of the future will demand complex and innovative audio systems that can't be designed by mere "code guys," because they don't understand the subtleties of audio design like he does. But unless he knows how to implement those ideas himself, he will be back behind the mixing board faster than you can say, "More vocals in the monitor please."

THE MUSICIAN. At another session, one well known figure within our community (who referred to himself as "The Almighty") addressed a room of more than 100 audio professionals as "musicians." I'm all for making music, but let's face it, music is only a small part of what goes into a game these days. Now, I understand that there are many people who represented themselves to their current employers as "computer experts" because they were wizards with the musician's equivalent to a writer's word processor, the MIDI sequencing program. I also understand that some people fancy themselves rock stars because they make a good living by orchestrating hundreds of little black boxes to do exactly what they want, when they want them to, and crank out the epic soundtrack to "Syncopation 2, The Aural Equivalent to Chinese Water Torture." However, most of us down

Continued on page 63.

Nation's Largest Motion Capture Service Studios

Now Open From Coast to Coast

• Plug-ins to Alias, 3D Studio MAX, Softimage, Lightwave, Maya • 3D Face Captures • Optical 10 Camera Systems • Animals of Varied Proportions • Real-Time Viewing of Character Animation • Multiple Person Captures

Los Angeles

PCS Performance Capture Studios

Refreshing Creativity • State of the Art Studio and Equipment • Large Volume, Multiple Actor/Animal Capture • Real Time Optical Capture • Simultaneous 3D Face and Full-body Captures

Services:

- *Motion editing*
- *Motion re-targeting*
- *Model attachment*
- *Software tools*
- *Cyberscan available*
- *Training*
- *Project management*
- *Consultation*
- *Portable system*

11872 LaGrange, Los Angeles, CA 90025 USA
310-704-3036 FAX 310-841-2076
www.performancecapture.com

New York

Modern Uprising Studios

Our services fill the gap between raw data and final animation! Our staff has experience capturing over 10,000 moves used for games, broadcast and feature films.

Services:

- *Motion editing*
- *Data re-scaling*
- *Skin attachment*
- *Data conversion*
- *Project consulting*

Facility:

- *5,800 square feet studio space*
- *Large capture space*
- *Portable system*

Brooklyn Navy Yard, NY USA
718-852-0811 FAX 718-858-2459
www.modernuprising.com

MotionAnalysis **MoCap Systems also used by these industry leaders:**

Acclaim Entertainment Brilliant Digital Entertainment CAPCOM Electronic Arts Eolith Gremlin Koei Konami Namco RARE Riverhillsoft SNK Sony Square

The Best in Game Development Technology!

Bink - the new true-color codec from the author of Smacker!

Bink is available and revolutionizing game videos just as Smacker did four years ago! Bink is a "better-than-MPEG-II" class codec. Yes, that's right - **better than DVD!** It produces higher quality than MPEG II and is up to three times faster than other software decoders. Now there is finally a true-color codec good enough for game developers.

Bink is a hybrid block-transform and wavelet codec that can encode your video using 16 different compression techniques (wavelet, DCT, motion compensation, a variety of vector quantizers, Smacker-style, etc). With all of these techniques in one codec, Bink can handle almost any type of video. Bink also includes a psycho-acoustic based audio codec that is capable of 8 to 1 perceptibly lossless compression, so your audio will sound as good as your video looks.

Bink is available now for Windows 95/NT (DOS and MacOS coming soon) and supports DirectDraw, DIBSections, DirectSound, and the Windows waveOut system. Bink uses the YUV colorspace, so it can use DirectDraw overlays for hardware color conversion and smooth scaling. Bink also includes a bunch of hand-optimized assembly YUV to RGB colorspace blitters, so you'll be able to access Bink's output in any RGB format you like. Bink is a high-end codec, and as such, it requires at least a Pentium/150 (Pentium/200 preferred). Bink really doesn't have a minimum CD requirement - 320x240 animations look great at 150 kps (1x CD) and 640x480's only need 450 kps (3x CD). Bink does need a reasonable video card that is capable of pumping out all those true-color pixels, though. The Bink SDK is very similiar to our popular Smacker SDK, so integration is easy, and upgrading from Smacker is easier still.

You're really going to love Bink - download the Bink Tools (or call for a demo CD) and try it on your videos today!

MPEG Layer-3 decompression support!

Miles 5.5 includes both Intel x87 and AMD 3DNow optimized MPEG Layer-3 decoders (patent rights licensed from Thomson and Fraunhofer). MP3 provides perceptually lossless compression at 11 to 1 - your sound files will be **more than ten times smaller** and will sound exactly the same! Miles supports both on-the-fly decompression (for minimal memory use), or pre-playback decompression (for minimal CPU hit). Miles can even MP3-compress DLS instrument data.

High-level 3D sound support including built-in RSX 3D technology!

The Miles high-level 3D sound API supports our built-in RSX 3D technology (acquired from Intel), fast 2D simulated, DirectSound3D, Dolby Surround, Creative's EAX, and Aureal's A3D (all selectable at run-time). Better still, you won't be tied to a particular low-level 3D API, so you can use the technology that makes your game sound best.

New digital sub-system! Per-sample reverb, input API with chat codec, faster mixing, filtering, and plug-in support! Replaces the DirectSound mixer!

Miles 5.5 has tons of new digital audio features: per-sample reverb, input support (including a low-bandwidth voice chat codec), on-the-fly filtering, and a new plug-in system for adding run-time sound effects which can be called at the pre- or post-mix stage. Miles also uses its own mixer even when running under DirectSound - if your game starts and stops multiple sound effects frequently, then this new feature can potentially double your frame rate!

Of course, Miles 5.5 provides all of the features you loved in earlier versions: complete digital mixing and format conversions, interactive MIDI playback with a built-in DLS software synthesizer, hard drive or CD-ROM digital streaming, red book CD-audio support, powerful callbacks, and much more!

Smacker - the best 256-color codec on the planet!

Smacker is still available and better than ever! Smacker is still the best codec for situations such as: 256 color games (of course), games targeting low-end machines (Pentium 133 and below), games that need video sprites or video transparency (which is much faster in 256 colors), cell-based (cartoon-style) videos, animated 3D texture compression, and very high-resolution videos (800x600 or 1024x768 - only Smacker is fast enough for videos this big).

The Smacker SDK is available for DOS, 16-bit Windows, Windows 95/ NT, Win32s, 68K Mac, and PowerMac. The SDK's API is identical across these platforms, and Smacker files can be played without conversion on each platform.

Smacker supports DirectDraw, DIBSections, WinG, DispDIB, SVGA VESA, and GWorlds (MacOS) for graphics output. For sound output, Smacker supports the Miles Sound System, DirectSound, the Windows waveOut system, and the Sound Manager (MacOS).

Who's using RAD? Everyone! Here's a partial list of our customers:

Activision, Blizzard, Eidos, Ion Storm, Microsoft, Electronic Arts, Bullfrog, LucasArts, Access, Bungie, Raven, DreamWorks, Humongous, Kesmai, Monolith, Origin, Sony, Psygnosis, Papyrus, Accolade, Acclaim, GT Interactive, Hasbro, Maxis, Sierra, Epic, Westwood, Take 2, The Learning Company, 989 Studios, Iguana, Rage Software, Probe, Cavedog, Midway, Vipah, Creative Wonders, New World Computing, 3DO, The Neverhood, Core Design, Blue Byte, Dynamix, Microprose, Viacom, Hypnotix, Impressions, Stormfront Studios, Virgin Interactive, Krisalis, Sir-Tech, Byron Preiss, Time Warner, Software Sorcery, McGraw Hill, Looking Glass, Adrenalin, Legend, Spectrum Holobyte, Strategic Simulations, Software 2000, Bethesda, Triloisyte, Sega, Compton's New Media, Firestorm, Criterion, Interactive Magic, SCi Limited, Adeline, Rocket Science, Atari, Velocity, Sculptured Software, Capcom, Software Toolworks, MECC, and Knowledge Adventure.

... over 2,300 games - see our web site for title lists!

Bink, the Miles Sound System, and Smacker are licensed on a per-product or per-site basis with NO ROYALTIES WHATSOEVER!

425-893-4300

335 Park Place Suite G109
Kirkland, WA 98033
425-893-4300
FAX 425-893-9111
sales@radgametools.com
www.radgametools.com

Made with love by

Our goal is to preserve classic video game magazines so that they are not lost permanently.

People interested in helping out in any capacity, please visit us at retromags.com.

No profit is made from these scans, nor do we offer anything available from the publishers themselves.

If you come across anyone selling releases from this site, please do not support them and do let us know.

Thank you!